Tralics, a LaTeX to XML translator; Part II

José Grimm

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Tralics, a LaTeX to XML translator Part II

José Grimm

N° 310

September 2005

Thème NUM

ISSN 0249-0803

INRIA
SOPHIA ANTIPOLIS

Tralics, a LaTeX to XML translator
Part II

José Grimm*

Thème NUM — Systèmes numériques
Projet Apics

Rapport technique n° 310 — September 2005 — 285 pages

Abstract: In this document we describe Tralics, a LaTeX to XML translator, and its application to the Raweb. There are two parts: the first part describes the translator, the second part the tools required for the Raweb.

This document has different chapters; we shall describe first how TeX can read an XML file and convert it to Pdf; in effect, we shall describe the xmltex, fotex and mathml packages, written by D. Carlisle et S. Rahtz, with some minor bug corrections and additions. We show how style sheets can be used to convert the XML source into XSL/FO or HTML, or even XML. Finally, we shall explain the Raweb DTD.

Key-words: Latex, XML, HTML, MathML, XSLT, PostScript, Pdf, stylesheet, formating

* Email: Jose.Grimm@sophia.inria.fr

Tralics, un traducteur de LaTeX vers XML
Partie II

Résumé : Dans cet rapport nous décrivons le logiciel Tralics, un traducteur de LaTeX vers XML, et son application au Raweb. La première partie de ce document décrit le traducteur lui-même, et la deuxième partie explique tous les outils nécessaires pour exploiter les fichiers XML.

Ce document contient plusieurs chapitres: on expliquera d'abord comment TeX peut interpréter du XML et produire du Pdf; il s'agit des packages xmltex, fotex, mathml2, écrits par D. Carlisle et S. Rahtz, avec quelques corrections et ajouts. On expliquera comment des feuilles de style permettent de convertir le XML en XSL/FO ou HTML, ou même en XML. Finalement, on expliquera la DTD Raweb.

Mots-clés : Latex, XML, HTML, MathML, XSLT, PostScript, Pdf, feuilles de style, formatage

Chapter 1

Introduction

This document is the second part of a report that explains Tralics and how it can be used for the Raweb. You will find a sequence of commented source files, most of them publicly available on the Web, followed by a large index (over twenty pages).

The index is sorted alphabetically; in the case of `<a>`, or `$c` or `\z@`, the first character is not used for sorting; the object `\@car` can be found with the character '@', before the letters; some private TeX commands have an at-sign in their name, sometimes in the middle, and sometimes at the start.

If you look at the letter Z, you can see five occurrences of the word 'year'. The first item uses a normal font; we use such a font for an attribute value (for instance, in the Raweb DTD, line 292, you can see that the `<citation>` element has an attribute from whose value can be 'year'). The second item uses a sans-serif font, this is the name of an attribute (for instance of the main Raweb element has a year attribute). Other items in the index use a type-writer font, and additional characters. For instance `<year>` is the name of a HTML or XML element, `$year` is the name of a xsl variable, and `\year` is the name of a TeX command (this one is not used here).

If you look at the letter I, you will find id, `id`, and `ID`. The case of `ID` is a bit special since it is the name of an attribute set (and one one such data structure is used). We use id for that name of an attribute, and `id` for an xsl template (whose purpose is to set the attribute). The index contains also entity names like `%list;`, such entities appear only in the DTD.

We do not index everything! Consider a source line, of the form

```
1371 \XMLNSAX{fo}{break-before}{\FObreakbefore}{auto}
```

The number in the left margin has nothing to do with the line number of the source document; it will be referenced to in some cases. This line is to be interpreted by TeX, it has 28 tokens. However, for a human reader, it contains five tokens, that could be indexed. The first token is the name of a TeX command that appears more than a hundred times (with and without the trailing X); it will not be indexed, as well as other common TeX commands like `\def`, `\gdef`, `\XMLelement`, or `<xsl:template>` in the part concerning style sheets, and `<!ELEMENT>` and `<!ATTLIST>` in the last chapter. The 'fo' in this example is a prefix that appears almost always after `\XMLNSA` or `\XMLNSAX`, thus it does not appear in the index. On the other hand, the attribute name, the associated TeX command and the default value will be found in the index. Consider a second example

```
1503 <xsl:variable name="Directory" select="concat($LeProjet,$year)"/>
```

Here, the line contains seven tokens. There is an element and two attributes. These will not be found in the index (the attribute name name will only be indexed if it appears in the HTML or XML document, not in the style sheet as xsl keyword). Xsl functions like 'concat' will not be indexed.

On the other hand you will find the three variables `$Directory`, `$LeProjet` and `$year` (in the line above, one variable is set, the other two variables are used).

The first chapter describes xmltex.tex. This is a TEX file whose purpose is to read and evaluate an XML file. The interpretation depends on some user commands, to be put in a .xmt file (the "user" here being the guy who designs the DTD of the XML file, as opposed to the author of the document, or the author of xmltex). The file contains a lot of commands of the form `\expandafter`, `\csname`, `\edef`, and the like, that are not described in standard LATEX books. If you understand this file, you can be called a TEX Master (according to the TEXbook, a Master is somebody who understand tables, a Grandmaster is somebody who can design output routines; the whole XML stuff described in this report is somewhere between these two levels). It is howver a challenge for a software like Tralics to be able to read the xmltex.tex file.

Using xmltex is easy. For instance, Chapter 3 explains how maths can be interpreted (this is an extension of the work of Carlisle, the author of xmltex). We have added commands that interpret the picture environment, and some extensions; the only diffilcult here is that the commands have an irregular syntax, so that the standrd mode of evaluation cannot be used (for instance, if you say

```
~~~~~~<oval x='1.2' y='3.4' specs='lt'> Text</oval>
```

we must call the associated TEX command like this

```
~~~~~~\oval(1.2,3.4)[lt]{Text}
```

rather than

```
~~~~~~\oval{1.2}{3.4}{lt}{Text}
```

Perhaps, the easiest way would be to write an intermediate command. The code is only given as an example of what can be done; it is not completely tested, and not used for the Raweb at all, because we do not know how to convert it into HTML.

Chapter 2

Interpreting XML in TeX

We shall describe here the xmltex.tex file. This is a piece of code written by David Carlisle as described in [1], it is a follow-up to typehtml, a package for typesetting HTML. The idea was to write a TeX file that interprets some XML code and typesets it, using rules defined in some other files (the so-called .xmt files), that depend on the DTD or namespaces. Some of these files are described in following chapters. This interpreter is used for the production of the Raweb (on figure 1 of the first part of this document, the arrows from 'xmlfo' to 'PDF' or 'PS' use this file). The XML file contains a lot of Unicode characters, that can be coded using iso-latin1 or UTF-8 encoding. Interpreting them in TeX is a real challenge. Here is an example:

```
<m:math overflow="scroll">
  <mrow xmlns="http://www.w3.org/1998/Math/MathML">
   <msup><mi>L</mi> <mn>2</mn> </msup><mo>&#x2192;</mo>
    <msup><mi>L</mi> <mi>&#x221E;</mi> </msup></mrow></m:math>
```

In this example, only ASCII characters are used, complicated things are written in the form `∞`, this is the same as `∞`. The `<mrow>` element has an xmlns attribute, it is hence the same as `<m:mrow>`. The action associated to this element is stored in some command, so one question is: what's the name of this command? every Unicode character is allowed, i.e., much more than the 256 internal characters of TeX. We have another problem, it is that an element name could be entered as `<José>` (iso-latin1 encoding). A good encoding is UTF-8, since it allows encoding of all Unicode characters on 8 bits. For instance, the representation of a character looks like this: `\8:Ã©`, and that of the element is `\E:0:JosÃ©`. Here between the two colons we have the value of the namespace, a sequence of digits, where 0 represents the empty namespace, 3, the MathML namespace, etc. (namespaces are defined in [10]). All commands that are dynamically created start with a prefix. This is `8:` for a UTF-8 character, `A:` for the global attribute list, `E:` for the start of an element, `E/:` for the end of an element, `Q:` for a processing instruction, like `<?xml?>`, `XML:` for a namespace. An example of such a namespace command is `\XML:http://www.w3.org/1998/Math/MathML`. Usual LaTeX commands contain only letters, reserved names may contain @; using a prefix with a character other than these reduces the risk of conflict with existing commands. We must use `\csname` for producing these commands. The math formula above is $L^2 \rightarrow L^\infty$.

In order to make the code easier to understand, we have invented some commands that are inlined in the real code (for efficiency reasons). The command `\XML:http.../MathML` (full name shown above) contains the unique identifier for the MathML namespace; this is in fact the number 3. It can be constructed via `\jg@NSuri`. In the case of `<m:math>`, the value of the 'm' prefix is the same number, it will be obtained via `\jg@namespace{m}`. In fact, when we parse an element, the prefix is in a global variable, so that we can use the parameterless command `\jg@this@namespace`.

```
\def\jg@NSuri#1{\csname XML:#1\endcsname}
\def\jg@namespace#1{\csname XMLNS@#1\endcsname}
\def\jg@this@namespace#1{\jg@namespace{\XML@this@prefix}}
```

In some cases, we need a canonical version of a string. We shall use the `\catxii` command for this: if `\val` is a command that expands to '`some/val`' then the expansion of '`\meaning\val`' is '`macro:->some/val`', this is a list of character tokens with category code 12 (except for spaces). The `\strip@prefix` command removes everything up to '`>`', it yields '`some/val`'. We need an `\expandafter` for changing the order of expansion.

```
\def\catxii#1{\expandafter\strip@prefix\meaning#1}
```

See the TEXbook [5] for details about `\expandafter`, category codes, the result of `\meaning`, what happens if `\meaning` produces no greater-than sign, etc. See the LATEX source code for `\strip@prefix`. See the Unicode book [7], paragraph 2.5, for the definition of encodings like UTF-8, UTF-16 and UTF-32, and the whole book for the significance of characters U+221E.

UTF-8 encoding is defined as follows. A character X will be represented using a variable number of bytes, say A, AB, ABC or ABCD. Let x be the integer value of X, and a, b, c and d the values of A, B, C, and D. The first byte indicates the length of the sequence: if $a < 128$, the sequence is of length one, and $x = a$. Otherwise, a starts with k bits 1 followed by a 0 bit, the sequence is of length k, the $k-1$ characters that follow start with a 1 and a 0 (and have 6 significant bits). These are the relations we shall use:

$$
\begin{array}{lll}
\text{If } 0 \le x < 2^7 & k=1 & x = x_1 \quad a = x \\
\text{If } 2^7 \le x < 2^{11} & k=2 & x = x_1 2^6 + x_2 \\
 & & a = 128 + 64 + x_1, b = 128 + x_2 \\
\text{If } 2^{11} \le x < 2^{16} & k=3 & x = x_1 2^{12} + x_2 2^6 + x_3 \\
 & & a = 128 + 64 + 32 + x_1, b = 128 + x_2, c = 128 + x_3 \\
\text{If } 2^{16} \le x < 2^{21} & k=4 & x = x_1 2^{18} + x_2 2^{12} + x_3 2^6 + x_4 \\
 & & a = 128 + 64 + 32 + 16 + x_1, b = 128 + x_2, c = 128 + x_3, d = 128 + x_4
\end{array}
$$

In all cases $0 \le x_i < 64$. The case $x \ge 2^{21}$ is not handled. As an example, the character with code 233 is coded as `Ã©`. Note: assume that X is an iso-latin1 character, if it fits on seven bits it is represented by itself. Otherwise, if $128 \le x < 128 + 64$, the first character is `Â`, the second is X, and if $x \ge 128 + 64$ the first character is `Ã`, the second has value $x - 64$ (note that most useful latin1 characters are in the range 192-255).

The file we consider starts like this:

```
1 %% Copyright 2000 David Carlisle, NAG Ltd.
2 %% re-released by Sebastian Rahtz June 2002
3 %% This file is distributed under the LaTeX Project Public License
4 %% (LPPL) as found at http://www.latex-project.org/lppl.txt
5 %% Either version 1.0, or at your option, any later version.
```

Unless told otherwise, newline characters are ignored in the xmltex file (in particular, on line 6, the space after the opening brace). More generally, lots of characters have category codes that depend on the context. We have not shown all these category changes; since definitions are in local groups, they are generally global (hence the `\gdef` here).

2.1 Constructing characters

Let's consider the following task: We have a character X, with code x, and x is in `\count@`. We want to find the bytes ABCD, with codes a, b, c and d. These quantities are obtained by writing x in base 64, with digits x_i, and we add 128 to everything. The first byte is a bit more complicated to compute. This piece of code uses two temporary registers `\@tempcnta` and `\@tempcntb` for the division. It replaces x by its quotient, and puts in the `\uccode` of `‘#1` the next byte (this assumes that the argument of the command is a character).

```
\gdef\XML@utfeight@a#1{
     \@tempcnta\count@
     \divide\count@64
     \@tempcntb\count@
     \multiply\count@64
     \advance\@tempcnta-\count@
     \advance\@tempcnta"80
     \uccode‘#1\@tempcnta
     \count@\@tempcntb}
```

This is the caller of the preceding command:

```
\gdef\XML@charref#1#2;{
  \begingroup
  \uppercase{\count@\if x\noexpand#1"\else#1\fi#2}\relax
  \ifnum\count@<"80\relax
    \uccode‘\~\count@
    \uppercase{
    \ifnum\catcode\count@=\active
      \gdef\XML@tempa{\utfeightay~}
    \else
      \gdef\XML@tempa{~}
    \fi}
  \else\ifnum\count@<"800\relax
     \XML@utfeight@a,
     \XML@utfeight@b C\utfeightb.,
  \else\ifnum\count@<"10000\relax
     \XML@utfeight@a;
     \XML@utfeight@a,
     \XML@utfeight@b E\utfeightc.{,;}
   \else
     \XML@utfeight@a;
     \XML@utfeight@a,
     \XML@utfeight@a!
     \XML@utfeight@b F\utfeightd.{!,;}
    \fi
    \fi
  \fi
  \endgroup}
```

There is a similar command, except that the test on lines 21-25 is assumed to be true, and code on line 22 is executed. It seems to be used only for reading auxiliary files in XML format; however, the .aux files contain currently no XML code.

```
\gdef\XML@charref@tex#1#2;{
  ...}
```

We shall see in the sequel some instances of 'black magic'. The result of '`\uppercase{xe9}`' is '`XE9`'. However, if you say '`\uppercase{\foo~}`' the result is '`\foo W`', where W is the character found in the uc table of the tilde character, and the category code of this character is the same as the tilde character (in general active). The substitution is done before `\foo` is evaluated[1]. In some cases, `\foo` is `\endgroup`. In our case, the group ends at line 41. The `\uppercase` on line 17 is not black magic. The idea is the following: imagine that we want to read something like '`é`' or '`é`', and that the ampersand and sharp characters have been read. Then `\XML@charref` reads all characters up to the semi colon. Arguments are 2, 33 in one case, x, e9 in the other case. A construction like `\count@="E9` puts 233 into `\count@`, upper case letters are needed. What `\uppercase` produces in our example is '`\count@\if X\noexpand X"\else X\fi E9`'; there is a `\relax` in the code whose purpose is to mark the end of the number (we do not want the `\ifnum` to be expanded before assignment is done); this `\relax` command could have been in the uppercase list. I don't know if `\noexpand` is needed here[2]. The effect of the conditional is just to replace the X by a `"` (you cannot do this using black magic, because the double quote has to be of category 12, so that the argument of the command must be of category code 12). What our code does is to put the number (say 233) into `\count@`. It chooses one of four alternatives on line 18, 26, 29 and 33; it corresponds to the number of bytes used to represent the Unicode character in UTF-8 format. In any case, the result is a definition of `\XML@tempa` as a command that start with `\UTF8?` (this is a shorthand for one of `\UTF8ax`, `\UTF8ay`, `\UTF8az`, `\UTF8b`, `\UTF8c`, or `\UTF8d`, the real name of the command is `\utfeightax`, etc.) followed by some characters (1, 1, 1, 2, 3, and 4 respectively).

Let's start with the case of one byte, lines 19-25. We have a special case here, because the UTF-8 character can be represented by a single TEX character; we use it, in the case where it is not expandable (i.e., is non active); the code on lines 41-42 does not use this simplification. As an example, if the number x is 65, then `\XML@tempa` will contain `A`; if it is 60, it will contain '`\utfeightay<`' (we assume that the less-than sign is of category code 12 when the code is read, this is needed on line 18, and of category 13 when the code is executed).

In the case where more than one byte is used, the idea is the following. We have to compute some integers a, b, c and d (two three or four values are required). These integers are in the range 1–255. If we store them in the uc-slot of A, B, C or D, then `\uppercase{ABCD}` will give a sequence of four characters, whose codes are the numbers a, b, c and c. Instead of these letters, point, exclamation point, comma and semi colon are used, in a random order. This is completely irrelevant since modifications are local (the group ends on line 41), and the `\uppercase` on line 47 sees only these character tokens, together with non-character tokens that are not affected. The code could be slight optimized if, on one hand, we notice that a is always stored in '.' (point) and, on the other hand, that b could always be stored in '!' (exclamation point). On lines 26 to 37 we compute b, c and d, and call `\XML@utfeight@b` with four arguments. Argument `#3` is the character that will hold a, argument `#4` is the list of characters that are already set, argument `#2` is command command name, one of the `\UTF8?` commands mentioned above. The first argument is a C, E, or F. Remember that $a = x_1 + s$, where x_1 is in `\count@`, s depends on the number of bytes. It is sixteen times 12, 14 or 15 (in base 16, it is C0, E0 or F0). What the next function does is then obvious:

```
44 \gdef\XML@utfeight@b#1#2#3#4{
45     \advance\count@"#10\relax
46     \uccode`#3\count@
47     \uppercase{\gdef\XML@tempa{#2#3#4}}}
```

Assume that our number is 233 (or E9, in base 16). We have $x_1 = 3$ and $x_2 = 41$. This gives $b = 128 + 41$, stored in the `\uccode` of `#4`. This is the character ©. Here `#1` is C, `"#10` is 192.

[1] The `\uppercase` command cannot be expanded; however its evaluation is a sequence of tokens that will be read again, expanded, and evaluated.

[2] The first character should be X or a digit, otherwise, the XML source is invalid. If you replace xe9 by xy9 or ye9, strange errors may be signaled; the case of a non-ASCII character is worse

Thus we store 195 (this is the code of Ã) in the `\uccode` of `#3`. Thus, the effect of the uppercase is to define the command `\XML@tempa` (this is temporary name that any name command may redefine), it takes no argument, expands to `\utfeightbÃ©`. The important point to remember: `\XML@charref` puts in `\XML@tempa` a list of tokens, this list is independent of the context, but the commands in the list have a meaning that depends on the context (redefined by the commands defined in the next paragraph).

2.2 Using UTF-8 characters

The piece of code that follows defines the six commands `\UTF8?` (there are other versions of the same commands). These definitions are useful in a context where we evaluate a piece of text.

```
\def\unprotect@utfeight{
  \let<\XML@lt@markup
  \let&\XML@amp@markup
  \def\utfeightax##1{
    \csname 8:\string##1\endcsname}
  \let\utfeightay\utfeightax
  \let\utfeightaz\utfeightax
  \def\utfeightb##1##2{
    \csname 8:##1\string##2\endcsname}
  \def\utfeightc##1##2##3{
    \csname 8:##1\string##2\string##3\endcsname}
  \def\utfeightd##1##2##3##4{
    \csname 8:##1\string##2\string##3\string##4\endcsname}}
```

For instance, `\utfeightbÃ©` expands to `\csname8:Ã\string ©\endcsname`. We shall see in a minute why all characters have to be protected, except the first one. If we expand this, we get the command with this strange name `\8:Ã©`. This command is assumed to typeset the Unicode character 233. Its definition could be, for instance, '`\ifmmode \acute{e}\else \'{e}\fi`'. Such a definition is valid only in a context where we typeset the object. Inside an `\edef`, the expansion of the conditional may give random results, inside a `\csname`, some tokens are illegal. Note that, in this command, less-than and ampersand are active, they scan something in the XML file; they should be input as '`<`' or '`&`' if you want a typeset < or &.

The next command looks funny:

```
\gdef\UnicodeCharacter#1#2{
   \begingroup
   \def\active{\catcode\count@}
   \XML@charref#1;
   \expandafter\expandafter\expandafter
   \expandafter\expandafter\expandafter
   \expandafter
    \gdef\XML@tempa{#2}
  \endgroup}
```

There are seven `\expandafter` in a row. Write `\E` instead, in order to gain space. Assume that we have a command `\A` that expands to `\B` that expands to `\C` that expands to `\D`. The expansion of '`\E\E\E\E\E\E\E\gdef\A`' is '`\E\E\E\gdef\B`'. This expands to '`\E\gdef\C`'. This expands to '`\gdef\D`'. Suppose that we say `\UnicodeCharacter{233}{\'e}`. In this case `\XML@charref` will define `\XML@tempa` as shown above. This is our `\A`. The expansion `\B` is `\utfeightbÃ©`. Its expansion `\C` is `\csname...`, its expansion `\D` is `\8:Ã©`. Hence, the code is `\def\8:Ã©{\'e}`. Thus, we know how to define every Unicode character. There is a little hack here (on line 63, you

see why?). Characters like A, B, C, typeset to themselves. But some other characters have to be defined. We say for instance

```
70 \UnicodeCharacter{94}{\textasciicircum}
71 \UnicodeCharacter{x5C}{\textbackslash}
72 \UnicodeCharacter{x5F}{\textunderscore}
73 \UnicodeCharacter{13}{ \ignorespaces}
74 \UnicodeCharacter{32}{ \ignorespaces}
75 \UnicodeCharacter{9}{ \ignorespaces}
```

These definitions come from the xmltex.tex file, and the Raweb redefines the character `x5C`, so as to allow it in math mode also. The definition of characters 9, 13 and 32 (spaces) is a bit strange: the `\ignorespaces` command expands the next token, and removes it, if it is a space; hence spaces given in the form ` ` are not removed. Worse: '`\parindent = 12 cm`' becomes illegal if what follows the equals sign comes from an XML file. The xmltex.tex file also has these definitions[3].

```
76 \expandafter\def\csname8:\string<\endcsname{\ifmmode\langle\else\textless\fi}
77 \expandafter\def\csname8:\string>\endcsname{\ifmmode\rangle\else\textgreater\fi}
78 \expandafter\def\csname8:\string{\endcsname{\{}
79 \expandafter\def\csname8:\string}\endcsname{\}}
```

What does the test on line 21 do? it compares the category code of `\count@` with `\active`; this is 13, and the test is false in the cases shown above (well, the backslash may be active while reading the XML file, it is surely not while processing line 71). Redefining `\active` has as side effect that it will expand to '`\catcode\count@`' and this is the same as `\catcode\count@`. As a consequence '`\XML@tempa`' expands to '`\utfeightay^`' that expands to `\csname...` that expands to `\8:^`. Hence, line 70 defines the command `\8:^`. This is what is desired. Note: when the XML file is read, all characters with code ≥ 128 are active, those with code ≤ 11 have category 12 (in fact, they are invalid in XML1.0).

The xmltex.tex file starts like this (before category codes of usual characters have been changed).

```
80 \count@0
81 \catcode0=13
82 \gdef\XML@tempa{
83  \begingroup
84    \uccode0\count@
85   \uppercase{\endgroup
86     \edef^^@{
87       \ifnum\catcode\count@=11 %
88         \noexpand\utfeightay\else\noexpand\utfeightax\fi
89       \noexpand^^@}
90     \expandafter\edef\csname 8:\string^^@\endcsname{\string^^@}}
91  \ifnum\count@<127\advance\count@1 \expandafter\XML@tempa\fi}
92 \XML@tempa
93 \catcode0=9
```

Here we have real magic. There is a loop over all numbers x between 0 and 127. The number x is in `\count@`. For each x, code on lines 83–90 are executed. The null character (number zero) is active, and its uc value is x. In lines 86–90, it will be replaced by the character x. Note that this character is input as `^^@`. Assume for instance that $x = 65$, so that it represents the letter A, or that $x = 61$ (character '='). The second `\edef` defines `\8:A` or `\8:=` to be A or = (note: the purpose of the `\edef` is to expand the `\string` in the body, so that the character in the body is a non-active character). Hence the effect is the same as `\UnicodeCharacter{65}{A}`.[4] The purpose

[3] These definitions of less-than and greater-than are wrong; they will be redefined later.

[4] The code on lines 70–79 overrides some of these settings

of the `\edef` on line 86 is the expansion of the conditional: we define A to be '`\utfeightay A`', and = to be '`\utfeightax=`'. The character after the command is active. Consider this:

```
\def\use@utfeightay{...}
\use@utfeightay ^^M ^_~%$#{}
```

We have simplified a bit the code. The idea is that, for the characters listed here, `\utfeightay` is used instead of `\utfeightax`. We shall see later that `\utfeightaz` is used for ampersand and less than in a case like `&` and `<`.

The following piece of code defines the commands `\UTF8?` (version two).

```
\def\utfeight@protect@internal{
  \let\utfeightax\noexpand
  \let\utfeightay\noexpand
  \def\utfeightaz{
    \noexpand\utfeightaz\noexpand}
  \let<\relax\let&\relax
  \def\utfeightb##1##2{
    \noexpand\utfeightb##1\string##2}
  \def\utfeightc##1##2##3{
    \noexpand\utfeightc##1\string##2\string##3}
  \def\utfeightd##1##2##3##4{
    \noexpand\utfeightd##1\string##2\string##3\string##4}}
```

What happens if a UTF8 character appears in an `\edef`? For instance, the character 'é', represented as '`\utfeightb Ãⓒ`' expands to the expansion of '`\noexpand\utfeightb Ã\stringⓒ`', namely '`\utfeightbÃⓒ`'. The only thing that might have changed is the category code of ⓒ. If it was active, it is now 12 (remember, the first character is never active). In the case `\utfeightay A`, the expansion is `A`, because `\utfeightay` is `\noexpand`. In the case of `\utfeightaz W`, the expansion is itself! Note that '`<`' and '`&`' are not modified.

This is version three:

```
\def\utfeight@protect@external{
  \def\utfeightax{
    \noexpand\noexpand\noexpand}
  \let\utfeightay\utfeighta@ref
  \let\utfeightaz\utfeighta@ref
  \edef<{\string<}
  \edef&{\string&}
  \def\utfeightb##1##2{
    ##1\string##2}
  \def\utfeightc##1##2##3{
    ##1\string##2\string##3}
  \def\utfeightd##1##2##3##4{
    ##1\string##2\string##3\string##4}}
```

In such a case, the expansion of '`\utfeightb Ãⓒ`' is '`Ãⓒ`' where both characters are of category code 12. This is very interesting in the case of `\write` that expands everything. The string Ãⓒ is the UTF-8 representation of é, and can be read again without trouble.[5] The expansion of '`\utfeightax~`' is '`\noexpand~`'. It will become `~` after another expansion. In the case of '`\utfeightax A`', the expansion is '`A`' because of the following lines:

```
\def\utfeighta@ref#1{
  \string&\string##\number\expandafter`\string#1\string;}
```

[5]The trouble is that, when the aux file contains `\newlabel`, its argument uses `\csname`, this expands to é, and this does not match the UTF-8 character.

Version four: this is the easy version: everything is converted into characters, of category code 12; in this case Unicode characters can be used inside a `\csname`.

```
\def\utfeight@protect@chars{
  \let\utfeightax\string
  \let\utfeightay\string
  \let\utfeightaz\string
  \def\utfeightb##1##2{
    ##1\string##2}
  \def\utfeightc##1##2##3{
    ##1\string##2\string##3}
  \def\utfeightd##1##2##3##4{
    ##1\string##2\string##3\string##4}}
```

2.3 Warnings

This piece of code is used in cases where we want to print something. It is the last definition of the `\UTF8?` series.

```
\def\utfeight@protect@typeout{
  \utfeight@protect@chars
  \let<\relax
  \let&\relax}
```

This is the piece of code that removes the traces.

```
\def\xmltraceoff{
  \global\let\XML@trace@warn\@gobble
  \global\let\XML@trace@warnNI\@gobble
  \global\let\XML@trace@warnE\@gobble
  \global\let\XML@attrib@trace\relax}
```

These are the commands that print a warning. We simplified a bit the code by removing (here) the body of some commands, and (elsewhere) calls to trace.

```
\def\XML@warnNI#1{
  {\let\protect\string\utfeight@protect@typeout\message{^^J#1}}}
\def\XML@warn#1{
  {\let\protect\string\utfeight@protect@typeout\message{^^J\XML@w@#1}}}
\def\XML@attrib@trace{...}
\def\XML@doattribute@warn#1#2#3{...}
\let\XML@trace@warn\XML@warn
\let\XML@trace@warnNI\XML@warnNI
\let\XML@trace@warnE\message
```

2.4 Reading the text

The next lines of code define a command `\nfss@catcodes`, such that, when executed, all characters have standard category codes. The @ character is a letter, quotes, less-than greater-than and equals-to are of category other.

```
 \def\nfss@catcodes{
  \catcode`\\0
  % Idem for {}%^@#"'<=>
  }
```

This changes even more category codes. Dollar, ampersand, hat, underscore, space have standard category codes, others have category 12.

```
\def\XML@reset{
  \nfss@catcodes
  % reset $&^_ space
  % reset :!=|
  \catcode`\~\active\def~{\nobreakspace{}}
  \let\XML@ns@a@\XML@ns@a@tex
  \let\XML@ns\XML@ns@tex}
```

The next lines of code define a command `\XML@catcodes`, such that, when executed, all characters have category codes useful for reading an XML file.

```
\def\XML@catcodes{
  \catcode`\ \active
  % same for: ^^M ^^I <>:[]%&"'=
  % same for: /!?-${}#_\~
  \def~{\utfeightay~}
  \let\XML@ns@a@\XML@ns@a@xml
  \let\XML@ns\XML@ns@xml
}
```

The following two commands are inlined for efficiency reasons. We have introduced them in order to gain space.

```
\def\Normalspace{\catcode`\^^I=10 \catcode`\^^M=10 \catcode`\ =10 }
\def\Activespace{\catcode`\^^I=13 \catcode`\^^M=13 \catcode`\ =13 }
```

This piece of code does a loop, starting with `\count@`, up to `\@tempcnta` (excluded). The loop puts the current number in the uc-code of tilde, and uppercasifies the value of `\XML@tempa`, to be defined later, in the form `\def\XML@tempa{{...}}`, double braces are needed because `\uppercase` want a brace-delimited list of tokens.

```
\gdef\utfeightloop{
  \uccode`\~\count@
  \expandafter\uppercase\XML@tempa
  \advance\count@\@ne
  \ifnum\count@<\@tempcnta
  \expandafter\utfeightloop
  \fi}
```

We leave it as an exercise to the reader to define a command `\XML@utfeight` whose expansion is '`utf-8`', all characters being of category code 12. This piece of code does nothing if the current encoding is '`utf-8`', otherwise it sets the current encoding to '`utf-8`', and does some action.

```
\gdef\XML@setutfeight{
  \ifx\XML@utfeight\XML@thisencoding
  \else
    \let\XML@thisencoding\XML@utfeight
    ...% see below
  \fi}
```

This is the action: for every character that is the first in a sequence of 2, 3 or 4 characters, it defines the character (for instance Ã) to take 1, 2 or 3 arguments. For instance Ã is defined as `\utfeightb Ã#1`. The first argument to `\utfeightb`, `\utfeightc`, or `\utfeightd` is not active! This works, because `\string~` is expanded to ~ of category code 12, where ~ is replaced by the `\uppercase` on line 174 by the character (for instance Ã), the `\utfeightb` command is not expanded since preceded by a `\noexpand`. The definition is in a double group (`\begingroup` on

line 185, braces on lines 188, 192, 194.) The definition is visible outside the group because it is global: we use `\xdef`. We could replace `\gdef` by `\def` here, whether the temporary is restored or not after the loop is irrelevant.

```
\begingroup
\count@"C2
\@tempcnta"E0
\gdef\XML@tempa{{\xdef~####1{\noexpand\utfeightb\string~####1}}}
\utfeightloop
\count@"E0
\@tempcnta"F0
\gdef\XML@tempa{{\xdef~####1####2{\noexpand\utfeightc\string~####1####2}}}
\utfeightloop
\@tempcnta"F4  \gdef\XML@tempa{{\xdef~####1####2####3{%
                          \noexpand\utfeightd\string~####1####2####3}}}
\utfeightloop
\endgroup
```

This defines a command named `\Q:xml`. It calls `\XML@xmldecl` after having changed the category code of white space.

```
\expandafter\gdef \csname Q:xml\endcsname{
  \Normalspace
  \XML@xmldecl}
```

This resets some category codes and calls `\XML@encoding`. The argument is something strange. The idea is that we parse `<?xml foo='bar'?>`. We read everything up to the end of the element, and provide a default encoding attribute. A `\relax` marker is put at the end.

```
\gdef\XML@xmldecl#1?>{
  \Activespace
  \XML@encoding#1 e="utf-8"\relax}
```

The XML norm (see for instance [8, 9, 3]) says (rules 23, 24, 32, 80) that in `<?xml?>` only the encoding attribute can start with the letter 'e'. This makes the loop easy. Note: other attributes are version, currently ignored (there are two versions of the XML standard, and the difference between them is tiny), and standalone (completely ignored).

```
\gdef\XML@encoding#1 #2{
  \if\noexpand#2e
    \expandafter\XML@encoding@aux
  \else
    \expandafter\XML@encoding
  \fi}
```

We shall see later that `\XML@quoted\foo` reads `'bar'` or `"bar"`, and calls `\foo` with the value 'bar'. The following piece of code grabs the attribute name, the equals sign, reads the attribute value and calls another command.

```
\gdef\XML@encoding@aux#1={
  \XML@quoted\XML@setenc}
```

Here the `\lowercase` is no magic: the XML norm says (rule 80) that all characters should be ASCII characters (letters, digits, dot, underscore, dash), and case independent. In the case where the encoding is not UTF-8, some file is read; for instance iso-8859-1.xmt. On page 6, we have seen how to find the UTF-8 representation of a latin1 character. It depends on whether the character is smaller or larger than 192. Two easy loops suffice to define all characters like é as `\utfeightbÃ©`.

```
\def\XML@setenc#1#2\relax{
  \lowercase{\gdef\XML@tempa{#1}}
```

```
214   \xdef\XML@tempa{\catxii\XML@tempa}
215   \ifx\XML@tempa\XML@thisencoding
216   \else
217     \ifx\XML@utfeight\XML@tempa
218       \XML@setutfeight
219     \else...% code not shown here
220     \fi
221   \fi}
```

2.5 Namespaces

You say `\XML@ns@alloc{foo}` in order to declare `foo` as a namespace name; after that the value can be found by `\jg@NSuri{foo}`. In the case this command is defined, there is nothing to do. Otherwise, we allocate a number using the counter `\XML@ns@count`, say 3, and put this in the command. We define two other commands: `\jg@namespace{3}` will be 3, an `\A:3` will be empty (we shall see that this is the global attribute list of the namespace). Note: we use here the pseudo commands `\jg@NSuri`, so that a double indirection is needed; as a consequence `\expandafter3` should be replaced by a sequence of three `\expandafter` tokens.

```
224 \def\XML@ns@alloc#1{
225   \expandafter3\ifx\jg@NSuri{#1}\relax
226     \global\advance\XML@ns@count\@ne
227     \expandafter3\xdef\jg@NSuri{#1}{\the\XML@ns@count}
228     \global\expandafter3\let\csname A:\the\XML@ns@count\endcsname\@empty
229     \expandafter3\xdef\jg@namespace{\the\XML@ns@count} {\the\XML@ns@count}
230   \fi}
```

The namespace stuff is initialized like this; number 0 corresponds to the empty namespace. Note that the recommendations say: The prefix 'xml' is by definition bound to the namespace name: http://www.w3.org/XML/1998/namespace.

```
231 \XML@ns@count-1
232 \XML@ns@alloc{}
233 \XML@ns@alloc{http://www.w3.org/1998/xml}
234 \def\XMLNS@xml{1}
235 \XML@ns@alloc{http://www.dcarlisle.demon.co.uk/xmltex}
```

The next piece of code is standard trick to convert 'foo:bar' into `{foo}{bar}` and 'foo' into `{}{foo}`. The auxiliary command sees 'bar' or '`\@`' as second argument, argument 3 is junk. This works only if `\\` does not appear in the argument, moreover, it is recommended that at most one colon appears, and no `\@`, otherwise, two many tokens are considered as junk.

```
236 \gdef\XML@ns@xml#1{\expandafter\XML@ns@a@xml#1:\@:\\}
237 \gdef\XML@ns@a@xml#1:#2:#3\\{
238   \ifx\@#2 \XML@ns@b{}{#1}
239   \else    \XML@ns@b{#1}{#2}
240   \fi}
```

The function above depends on the category code of the colon character. We define an alternative version of the command and its helper, and install `\XML@ns@a@` to be the TEX variant, but it may be redefined (see lines 160 and 167).

```
241 \def\XML@ns@tex#1{...}
242 \def\XML@ns@a@tex#1:#2:#3\\{...}
243 \let\XML@ns@a@\XML@ns@a@tex
244 \let\XML@ns\XML@ns@tex
```

What this code does is just to expand everything (in order to get a canonical form). Thus `\XML@ns`, as well as all its variants, take a sequence like '`foo:bar`', puts it in a canonical form, and puts '`foo`' in `\XML@this@prefix`, '`bar`' in `\XML@this@local`.

```
\def\XML@ns@b#1#2{
  \begingroup
  \utfeight@protect@chars
  \xdef\XML@tempa{#1}
  \xdef\XML@tempb{#2}
  \endgroup
  \let\XML@this@prefix\XML@tempa
  \let\XML@this@local\XML@tempb
  }
```

2.6 Redefining `\protect`

In order to prevent premature expansion, you can insert `\protect` before a command; this makes it "robust"; the `\protect` command is defined in LaTeX, its value depends on the context. It may be `\@unexpandable@protect`, that is `\noexpand\protect\noexpand`. Hence `\protect\foo` expands to itself in an `\edef`. On line 96, we define a command so that `\utfeightb ÃC` (the internal representation of é) also expands to itself. In this section, we modify all context switch commands in order to make all UTF-8 characters naturally robust.

We start with a modified `\xdef` in which `\protect` and UTF-8 characters are left unchanged. This works well in a group because the end of the group restores the old value.

```
\def\unrestored@protected@xdef{
   \utfeight@protect@internal
   \let\protect\@unexpandable@protect
   \xdef
}
```

Another extension to LaTeX: Here everything is done in a group, the definition is global, the modifications to `\protect` and `\UTF8?` are local; the group is terminated after the `\xdef` because of the `\afterassignment`.

```
\def\protected@xdef{
   \begingroup
   \utfeight@protect@internal
   \let\protect\@unexpandable@protect
   \afterassignment\endgroup
   \xdef}
```

Yet another one: No group is used here, and `\afterassignment` gets another token as argument. This is useful if we do not want an `\xdef` (for instance, `\refstepcounter` uses this to define the current label). The meaning of UTF-8 characters is not restored, but reset to XML mode.

```
\def\protected@edef{
   \let\@@protect\protect
   \let\protect\@unexpandable@protect
   \utfeight@protect@internal
   \afterassignment\restore@protect
   \edef
}
```

We have to restore `\protect` and some other commands.

```
\def\restore@protect{\let\protect\@@protect
   \unprotect@utfeight}
```

We have to redefine `\protected@write`. This is a command that takes 3 arguments. It writes the last argument on the file defined by the first argument. Protection works as follows: there is an `\edef` that will expand all tokens but the protected ones, the current page reference (i.e., `\thepage`), including side-effects that come from evaluating the second argument. For instance, in Chapter 4, line 1912, there is an example where `\jgFOlabel` is set to `\relax`; the `\addtocontents` command defines `\label`, `\index` and `\glossary` to gobble their arguments.

```
\long\def \protected@write#1#2#3{
      \begingroup
       \let\thepage\relax
       #2
       \utfeight@protect@external
       \let\protect\@unexpandable@protect
       \edef\reserved@a{\write#1{#3}}
       \reserved@a
      \endgroup
      \if@nobreak\ifvmode\nobreak\fi\fi
}
```

We must also redefine this (it is used by `\typeout`).

```
\def\set@display@protect{
  \let\protect\string
  \utfeight@protect@typeout}
```

2.7 The catalogue

The catalogue is an association list, a sequence of the form `\key{val1}{val2}`. We have mentioned elsewhere that adding something at the end of a token list is not obvious. Here we proceed as follows. Consider

```
\edef\val{\noexpand\the\list\noexpand\key{\catxii\val}}
```

If we assume that `\list` is a command that cannot be expanded and `\val` expands to '`some/val`', the code above puts `\the\list\key{some/val}` into `\val`. Assume now that `\list` is a reference to a token list, and that we say

```
\list\expandafter\expandafter\expandafter{\val{aux}}
```

Since `\list` is a reference to a token list, the code above is an assignment, after `\list` we have a token list, and the first token is expanded to see if it is a left brace. Because of the `\expandafter` the code is equivalent to

```
\list\expandafter{\the\list\key{some/val}{aux}}
```

Now, the token that follows `\list` can be expanded; hence the result is the same as

```
\list{<value of the list>\key{some/val}{aux}}
```

We can also say something like

```
\list\expandafter{\the\expandafter\list\expandafter\key\val{aux}}
```

Here the effect of `\expandafter` is to expand `\the`; this expands the token that follows, namely the `\expandafter`, so that `\val` is expanded. This the result is the same as

```
\list{<value of the list>\key some/val{aux}}
```

The catalogue is a token list defined by sequence of assignments like this:

```
\SYSTEM   {http://www.oucs.ox.ac.uk/dtds/tei-oucs.dtd} {tei.xmt}
\NAMESPACE{http://www.w3.org/1998/Math/MathML}         {mathml2.xmt}
\NAMESPACE{http://www.dcarlisle.demon.co.uk/sec}       {sec.xmt}
\NAME{langtest}                                        {langtest.xmt}
\NAME{TEI.2} {tei.xmt}
\NAME{html}                                    {html.xmt}
\NAMESPACE{http://www.w3.org/1999/XSL/Format}           {fotex.xmt}
```

Here the last item on each line is the name of a TEX file to load in some cases. There are five different items in the catalogue, thus five commands that put things in it, and five other commands that extract something. The action of `\FOO{A}{B}` is essentially to add `\XML@@FOO{A}{B}` at the end of the token list.

Let's start with the `\PUBLIC` command. It takes two arguments, an URI and a file name.

```
\def\PUBLIC#1#2{
 \xdef\XML@tempa{#1}
 \xdef\XML@tempa{\noexpand\the\XML@catalogue\noexpand\XML@@PUBLIC
             {\catxii\XML@tempa}}
 \global\XML@catalogue\expandafter\expandafter\expandafter{
   \XML@tempa{#2}}}
```

Same idea here.

```
\def\SYSTEM#1#2{
 \xdef\XML@tempa{#1}
 \xdef\XML@tempa{\noexpand\the\XML@catalogue\noexpand\XML@@SYSTEM
             {\catxii\XML@tempa}}
 \global\XML@catalogue\expandafter\expandafter\expandafter{
   \XML@tempa{#2}}}
```

In the case of a namespace, for instance MathML, we compute the namespace number of it, and the catalogue associates to this number the file in which everything is defined.

```
\def\NAMESPACE#1#2{
  \utfeight@protect@chars
  \XML@ns@alloc{#1}
  \edef\@tempa{{\jg@NSuri{#1}}}
  \global\XML@catalogue\expandafter{\the\expandafter\XML@catalogue
     \expandafter\XML@@NAMESPACE\@tempa{#2}}
  \unprotect@utfeight}
```

This is the easiest of all commands, since we do not have to do anything with the arguments.

```
\def\NAME#1#2{
 \global\XML@catalogue\expandafter{\the\XML@catalogue\XML@@NAME{#1}{#2}}}
```

You run the catalogue by evaluating it. For instance, if you put '`foo`' into `\XML@PUBLIC`, then the value associated to `foo` by the `\PUBLIC` command will be put in `\XML@use`.

```
\def\XML@@PUBLIC#1#2{
  \gdef\XML@tempa{#1}
  \ifx\XML@tempa\XML@PUBLIC \def\XML@use{#2}\fi}
```

Same action for `SYSTEM`, `NAME` and `NAMESPACE`. The temporary variable has a different name.

```
\def\XML@@SYSTEM#1#2{
  \def\@tempa{#1}
  \ifx\@tempa\XML@SYSTEM \def\XML@use{#2}\fi}
```

```
\def\XML@@NAMESPACE#1#2{
  \def\@tempa{#1}
  \ifx\@tempa\XML@NAMESPACE  \def\XML@use{#2}\fi}
\def\XML@@NAME#1#2{
  \def\@tempa{#1}
  \ifx\@tempa\XML@NAME  \def\XML@use{#2}\fi}
```

You say `\XMLNS{html}{http://www.w3.org/1999/xhtml}`. The effect is to associate to the name `html` the namespace value of the second argument, for instance 17.

```
\def\XMLNS#1#2{
  \utfeight@protect@chars
  \XML@ns@alloc{#2}
  \edef\@tempa{{#1}{\jg@NSuri{#2}}}
  \global\XML@catalogue\expandafter{\the\expandafter\XML@catalogue
     \expandafter\XML@@XMLNS\@tempa}
  \unprotect@utfeight}
```

This piece of code is a bit strange; it might produce unexpected results. The idea is the following. The command `\XML@checkknown` will run the catalogue in case of unknown elements. In the case of `<TEI.2>` or `<html>`, where no namespace prefix is given, the command `\XML@NAME` is set, and `\XML@@NAME` may define `\XML@use`. However, if `\XMLNS` has been defined as above, this piece of code is also executed: it defines a default namespace, in particular it could replace `<0:html>` by `<17:html>`. The last action is to define `\XML@NAMESPACE`, and we are ready to run the catalogue again.

```
\def\XML@@XMLNS#1#2{
  \def\@tempa{#1}
  \ifx\@tempa\XML@NAME
    \edef\XMLNS@{#2}
    \edef\XML@this@element{\XMLNS@\noexpand:\XML@this@local}
    \let\XML@NAMESPACE\XMLNS@
  \fi}
```

2.8 Reading elements

Let's start slowly. This piece of code is executed whenever we see a less-than sign, that is the start of an element. We have to distinguish between `</foo>`, `<?foo>`, `<!foo>` and `<foo>`. The procedure reads one character. What makes everything interesting is that `\fi` tokens are missing. On the other hand, we have inserted a `\@` marker, whose purpose is to skip easily over all useless tokens.

```
\def\XML@lt@markup#1{
  \Normalspace
  \ifx/#1\XML@getend
  \else\ifx!#1\XML@getdecl
  \else\ifx?#1\XML@getpi
  \else\XML@getname#1\@}
```

The function that follows is defined in an environment where space, newline, and tabulation are active characters (remember that `\endlinechar` is -1, so that newline characters are produced only via `^^M`). The code makes these characters active, and defines them; this action is local to a group (the groups ends on line 352). When we typeset some text, it is wise to activate these characters; on the other hand, spaces have normal category code when scanning attributes. In any case, space characters disappear at end of line, this explains the need of the `%` signs here. This piece of code is called when the XML file contains `<foo>`; we have read the less than sign, and the

letter that follows; the letter is in `#1` (if you look at line 340, you see that `#1` is nothing else than the first argument of `\XML@lt@markup`, because this cannot be `\@`. It could be `Ã`, i.e., the first byte of a Unicode character). We know that this is not slash, not an exclamation point, not a question mark, and we close these conditionals. The reader should take some time, in order to understand how `\XML@tempa` is defined.

```
\gdef\XML@getname#1\@{
\fi\fi\fi
\begingroup
\Activespace
\def {\iffalse{\fi}\XML@getname@}
\let^^M %
\let^^I %
\def/{\iffalse{\fi}\XML@getname@/}
\def>{\iffalse{\fi}\XML@getname@>}
\unrestored@protected@xdef\XML@tempa{\iffalse}\fi#1}
```

The last line contains `\unrestored@protected@xdef`; this is a command that modifies the behavior of some UTF-8 characters; it assumes to be in a group (thus the 'unrestored'); it evaluates to `\xdef` (see line 255). After TEX has seen the opening brace, all tokens are expanded; as a result the '`\iffalse}\fi`' is ignored; note that the '`\iffalse{\fi`' that appears in the definition of space, tabulation, newline, slash, greater-than sign disappears also; the full expansion of these commands is: a closing brace, `\XML@getname@`, and maybe one character. It is this closing brace that terminates the `\xdef`; said otherwise, `\XML@tempa` will contain everything up to these characters. In the case `<foo>`, `<foo/>`, `<foo a='b'>`, it will contain '`foo`'. In the case of `<José>`, in a document with latin1 encoding, it will contain '`JosÃ©`', where the `Ã` is an active character.

The `\XML@getname@` is defined below. It closes the group in which space and other characters have a funny definition. The `\XML@begingroup` is a hack that saves some stack space. It has the same features as `\begingroup`. The `\XML@w@` command contains N spaces (where N is the current level). It is used for debugging, and argument grabbing. What the code does is: Put in `\begintag` the name of the element, in `\XML@parent` the current element (the parent of this one), initialize the current attribute list `\XML@attribute@toks` to the empty list, and parse the attributes.

```
\def\XML@getname@{
  \endgroup
  \XML@begingroup
  \edef\XML@w@{ \XML@w@}
  \let\begintag\XML@tempa
  \let\XML@parent\XML@this@element
  \XML@attribute@toks{}
  \XML@getattrib}
```

All these `\expandafter` in the code have as purpose to pop the conditional stack (said otherwise, if the command takes an argument, the argument will be what follows the `\fi`, not the `\fi` itself).[6] There are three cases to consider: there is an attribute, or there is none. In the case where there is no attribute, there are two subcases: the element can be empty or not. If you say `<foo␣␣bar='gee'>`, the first space was active, and read by the magic above; the second one has category code 10, and is discarded because the argument to this command is not a delimited argument.

```
\def\XML@getattrib#1{
  \ifx#1/
    \expandafter\XML@endempty
  \else
```

[6] Guess: why is there no `\expandafter` after the `\else`?

```
363   \ifx#1>
364      \expandafter\expandafter\expandafter\XML@startelement
365   \else
366     \XML@getattrib@a#1
367   \fi
368   \fi}
369 \let\XML@@getattrib\XML@getattrib
```

In the case of `<foo bar='1'/>`, when the slash is seen, the greater sign is read, and `</foo>` is pushed back in the input stream. After that, we proceed as if there were no slash. This means that `<foo/>` is the same as `<foo></foo>`.

```
370 \def\XML@endempty#1>{
371   \expandafter\XML@startelement
372   \expandafter<\expandafter/\begintag>}
```

Here is a little trick: in the case where we are reading `<foo bar='1'>`, line 366 contains the command `\XML@getattrib@a`, followed by the letter b, followed by `\fi\fi`, followed by `ar='1'` (still unread). What we do is a trick to read an optional space before the equals sign (the space after the equals sign disappears because `\XML@quoted` uses an undelimited argument; a space in the attribute value will not disappear, because `\XML@qq` uses a delimited argument). We save the attribute name in a variable, and read the value.

```
373 \gdef\XML@getattrib@a#1\fi\fi#2={
374   \fi\fi
375    \XML@set@this@attribute#1#2 \@
376    \XML@quoted\XML@attribval}
377
378 \def\XML@set@this@attribute#1 #2\@{
379   \def\XML@this@attribute{#1}}
```

You say `\XML@quoted\foo'bar'` or `\XML@quoted\foo"bar"`. In both cases, `\foo` is called with `bar` as argument. In general, error handling is very poor. The purpose of `\ERROR` here is not to report an error in the case of wrong syntax. It will be used on line 559.

```
380 \def\XML@quoted#1#2{
381    \ifx#2"\expandafter\XML@qq
382    \else\ifx#2'\expandafter\expandafter\expandafter\XML@q
383    \else
384      \ERROR#2
385    \fi \fi #1}
386 \def\XML@qq#1#2"{#1{#2}}
387 \def\XML@q#1#2'{#1{#2}}
```

In order to make things easier to understand, write `\Att` instead of `\XML@this@attribute`, this is the attribute to analyze. Write `\AL` instead of `\XML@attribute@toks`, this is the resulting list to which tokens will be added. Before we forget it: this command terminates on line 407, with `\XML@getattrib`, hence continues parsing the attribute list. The normalized attribute value is compared to `\XML@ns@decl`, a command that contains '`xmlns`' with category codes 12. If the attribute name is '`xmlns`', this defines the default namespace, if the attribute is '`xmlns:foo`', this defines the namespace prefix 'foo' for this element and its content. In both cases, we call `\XML@ns@uri`. Otherwise if the attribute is `foo:bar='gee'` we add `\XML@doattribute{foo}{bar}{gee}` to the token list.

```
388 \def\XML@attribval#1{
389   \xdef\XML@tempa{\catxii\XML@this@attribute}
390   \ifx\XML@tempa\XML@ns@decl
391     \XML@ns@uri{}{#1}
```

```
392     \else
393       \XML@ns\XML@this@attribute
394       \edef\XML@this@prefix{\catxii\XML@this@prefix}
395       \ifx\XML@this@prefix\XML@ns@decl
396         \XML@ns@uri\XML@this@local{#1}
397       \else
398        \begingroup
399        \utfeight@protect@internal
400         \xdef\XML@tempa{
401            \the\XML@attribute@toks
402            \noexpand\XML@doattribute{\XML@this@prefix}{\XML@this@local}{#1}}
403         \endgroup
404         \XML@attribute@toks\expandafter{\XML@tempa}
405        \fi
406      \fi
407   \XML@getattrib}
```

Assume that we have `xmlns:foo='http://www.w3.org/1998/Math/MathML'`. This piece of code allocates a number for the URI if not already done. Let's assume that this number is 3. It then defines `\XMLNS@foo` to be 3. In the case `xmlns='...'` it defines `\XMLNS@`, the default namespace.

```
408 \def\XML@ns@uri#1#2{
409   \utfeight@protect@chars
410   \XML@ns@alloc{#2}
411   \expandafter3\edef\jg@namespace{#1}{\jg@NSuri{#2}}
412   \unprotect@utfeight}
```

The next macro is called when we see the greater-than sign that closes the opening tag of an element. We first do something with default attributes, this will be explained later. After that, we split the element name into '`foo:bar`'. Assume that the namespace number of '`foo`' is 4, we put in '`\XML@this@element`' the tokens '`4:bar`'. We execute a piece of code that can possibly load a file in which the element's behavior is defined, and then, we execute the associated code. We shall see later that there is a command `\xmlgrab` that reads everything (including subelements) up to some end tag. This command redefines `\XML@doelement`. This explains why `\XML@doelement` is not inlined.

```
413 \gdef\XML@startelement{
414   \XML@default@attributes
415   \Activespace
416   \XML@ns\begintag
417   \edef\XML@this@element{
418     \jg@namespace{\XML@this@prefix\expandafter}\noexpand:\XML@this@local}
419   \XML@checkknown
420   \XML@doelement}
```

The action associated to `<m:math>` is just to call `\E:3:math`. We shall see later how this command can be defined.

```
421 \def\XML@doelement{
422   \csname E:\XML@this@element \endcsname}
```

Assume that we want to evaluate `\E:3:math`. This routine does nothing if the command exists. Otherwise, it "runs the catalogue", and does a check: a warning is printed in case where the command does not exists. Assume first that the prefix is empty (has number 0). In this case, the catalogue can have an entry for the name (defined via the `\NAME` command), defining a file to load. Otherwise, the catalogue should have an entry for the namespace (here 3) defined via

`\NAMESPACE`. In any case, the catalogue should define `\XML@use` the name of a file to be loaded. We simplified a bit the code by introducing the `\jg@this@namespace` command; note that we could use `\XML@this@element`. Important note: on line 353, there is a command `\XML@begingroup`, so that all definitions from the included file are local to this group. If the current element is not the root element, and if you want the definitions to apply to all elements (and not only the descendants of this one), the definitions should better be `\global`.

```
\def\XML@checkknown{
  \expandafter\ifx
    \csname E:\jg@this@namespace:\XML@this@local\endcsname
    \relax
   \let\XML@use\@empty
   \ifnum0=\jg@this@namespace
     \let\XML@NAME\XML@this@local
     \the\XML@catalogue
   \else
     \edef\XML@NAMESPACE{\jg@this@namespace}
     \fi
   \let\XML@NAME\relax
   \the\XML@catalogue
   \inputonce\XML@use
   \expandafter\ifx\csname E:\jg@this@namespace\csname
       :\XML@this@local\endcsname\relax
     \XML@trace@warnE{Undefined}
   \fi
 \fi}
```

2.9 End of element

Look at lines 337-340. When we are reading `</foo >`, the `\XML@getend` command is called after the slash has been read. This little piece of code grabs all tokens, until the end of the command (argument `#1`, unused), and everything up to the greater-than sign (argument `#2`). It closes the conditional, and calls another command (the purpose of the call is to get rid of the final space). Why this changes the category code of space and not tabulation is beyond me.

```
\def\XML@getend#1\@#2>{
  \fi
  \catcode`\ \active
  \XML@getend@a#2 \@}
```

We have now read `</foo >`, and `\endtag` contains '`foo`'. We extract the namespace part, and call a command (that may be redefined in case of grab).

```
\gdef\XML@getend@a#1 #2\@{
  \Activespace
  \def\endtag{#1}
  \XML@ns\endtag
  \XML@doend}
```

The action associated to `</m:math>` is to call the command named `\E/:3:math`. After that, we have to close a group (opened on line 353).

```
\gdef\XML@doend{
  \csname E/:\jg@this@namespace:\XML@this@local  \endcsname
```

```
453   \XML@endgroup
454   \Activespace}
```

2.10 Using attributes

Consider `<X:elt foo:bar='gee' color='red' xmlns:X='myX'/>`. We have already seen that the effect of the xmlns:X attribute is to define X as a namespace for this element and its children. In the case foo:bar, the definition of 'foo' could come later. For this reason, when we parse the attribute list, we construct a list of the form `\do{a}{b}{c}`,[7] in our case it contains `\do{foo}{bar}{gee}` and `\do{}{color}{red}`. This list is in `\XML@attribute@toks`. There is another list that contains terms of the form `\Att name\relax\cmd{val}action`[8], it depends on the behavior of the element `<X:elt>`. Imagine that any element in the `X` namespace has an attribute some:background, with a default value of black, and `<X:elt>` has an attribute color, with some default value, and that some action is associated to it. This second list is the argument of the macro whose definition follows. It is constructed by `\XMLelement`; this construction can occur because of autoloading of some package, and this depends on the current namespace, i.e., `myX`. What follows `\Att` is the name with its namespace, for instance `25:background` or `0:color`; it is followed by `\relax` and the name of the command in which the element can get the value; it is followed by the default value (in braces) and an action (a sequence of commands, generally empty). Let's assume that there is no action for the background, but `\checkcolor` for the color.

This piece of code evaluates both lists, with a double `\relax` at the end. We have to evaluate all namespaces, but there is no default namespace for attributes. For this reason we set `\XMLNS@` to 0.

```
455 \def\XML@setattributes#1{
456   \let\XMLNS@@\XMLNS@
457    \def\XMLNS@{0}
458    \the\expandafter\XML@attribute@toks#1\relax\relax
459    \let\XMLNS@\XMLNS@@}
```

Let's assume that our big list is the following `\do {foo} {bar} {gee} \do {} {color} {red} \Att 25:background\relax \setbg {black} 0:color\relax \setcol {blue} \checkcolor \relax \relax`. The command that follows evaluates the `\do`. It reads the three tokens that follow and constructs a command that will be used twice. If this command is `\T`, then `\T\foo` applies `\foo` to some argument, namely `\Att name\relax`, where name is the full name, for instance `17:bar` or `0:color`.

```
460 \def\XML@doattribute#1#2#3{
461   \xdef\XML@tempa##1{\noexpand##1{
462        \noexpand\XML@attrib\jg@namespace{#1}:#2\relax}}
463   \XML@tempa\XML@attrib@x{#3}
464   \XML@tempa\XML@attrib@y}
```

The command that follows is called twice in our example, with arguments '`\Att17:bar\relax`' and '`gee`', then '`\Att0:color\relax`' and '`red`'. The action is to define a command `\XML@tempb`, whose action is to read some tokens, and define some command to be '`gee`' or '`red`'. We shall see in a moment how this command is used. Putting a `\def` in a `\def` in a `\def` is unusual. There is a priori no reason why the second one should be global. The inner one has to be local.

```
465 \def\XML@attrib@x#1#2{
466     \gdef\XML@tempb##1#1##2##3##4\relax\relax{
```

[7] Here `\do` is short for `\XML@doattribute`
[8] Here `\Att` is short for `\XML@attrib`

```
467     \def##2{#2}
468     ##1##4\relax\relax}}
```

The command that follows is simple (its body has one line) but a bit subtle. Remember the long list of tokens shown above. It is of the form `\do...\do...\Att...\Att...\relax\relax`. We have read the `\do...`, and constructed a `\Att...`, which is in `#1`. Everything else is in `#2`. We apply `\XML@tempb` to the list, where the first `\do...` is removed and a new `\Att...` is added. Remember that `\Att` is followed by a name, then `\relax`, a command, a value, and maybe action. In `#1` we have only the name and `\relax`; we provide here `\XML@temp@l` as command name, and 6 as value. We show the code, then the explanations.

```
469 \def\XML@attrib@y#1#2\relax\relax{
470   \XML@tempb#2#1\XML@temp@l{6}\relax\relax}
```

Consider first the case where `#1` is '`\Att17:bar\relax`'. The command `\XML@tempb` takes four arguments, the first argument is delimited by `#1`, and the last by a double `\relax`. Our element `<X:elt>` knows nothing about 'foo:bar', so that the `#1` is the one provided on line 470. Thus, the first argument is `#2`, the second argument is `\XML@temp@l`, the third argument is '6', the last is empty. The effect of line 467 is to define `\XML@temp@l`, this is a dummy command, its definition is irrelevant. The effect of line 468 is to evaluate the long list again.

Consider now the case where `#1` is '`\Att0:color\relax`'. This is found in the long list because the element accepts the color attribute. Hence the arguments of `\XML@tempb` are the following: The first argument is '`\Att 25:background\relax \setbg {black}`' (in general, it starts with all the unhandled `\do...` commands), the second argument is '`\setcol`', the third is '`blue`', the last is '`\checkcolor \Att 0:color\relax \XML@temp@l {6}`'. The concatenation of arguments 1 and 4 is the long list, with the `\Att...` of color removed, and re-inserted at the end, with `\XML@temp@l` as command and 6 as value. The action is to define `\setcol` to '`red`'. The definition is local to the group started on line 353, ended on line 453. The action associated to `<X:elt>`, `</X:elt>` and descendants can see this value.

After all these `\do...` have been evaluated, our long list reduces to '`\Att 25:background\relax \setbg {black}` `\checkcolor \Att 0:color\relax \XML@temp@l {6} \relax \relax`'. This is the list constructed by `\XMLelement`, possibly re-ordered, where the command name associated to attributes that have a value is replaced by a dummy name. Note the placement of `\checkcolor` is this list: when it is evaluated, `\setcol` is defined, either to the value of the XML file or the default value. The definition of `\Att` is given here: it sets `\setbg` to black, and `\XML@temp@l` to 6. The only subtlety is that, if the default value is `\inherit` nothing happens. Note: the initial value should always be defined; however, the code checks this, replacing undefined by `\relax`.

```
471 \def\XML@attrib#1\relax#2#3{
472   \ifx\inherit#3\relax% #3 might be empty
473     \ifx#2\@undefined
474       \def#2{\relax}
475     \fi
476   \else
477     \def#2{#3}
478   \fi}
479 %
480 \let\inherit\XML@attrib % just some random name
```

2.11 Processing instructions

We consider here parsing `<?xml?>`. The layout here is awful, for the same reason as `\XML@getname`. We use however a different trick: we use `\csname` and define the delimiters (space, question mark), to evaluate to `\endcsname` (no namespace hacking needed here).

```
\gdef\XML@getpi#1\@{
\fi\fi\fi
\begingroup
\utfeight@protect@chars
\Activespace
\def?{\endcsname?}
\let \endcsname
\let^^M\endcsname
\let^^I\endcsname
\expandafter\XML@getpi@\csname
Q:}
```

Hence, in the case `<?xml something?>`, the `\XML@getpi@` command is called, with as argument the token `\Q:xml`. What we do is close the current group, activate spaces, evaluate the command. If the command is not defined, we put a `\XML@getpi@x` before it. Note that, if a command is constructed by `\csname`, its value is `\relax` instead of undefined; this is a local assignment, after `\endgroup`, the undefined value is restored. Note that the action associated to `<?xml?>` is defined on line 198.

```
\def\XML@getpi@#1{
  \endgroup
  \Activespace
  \ifx#1\@undefined
    \expandafter\XML@getpi@x
  \fi
  #1}
```

In the case where `<?foo something?>` is seen, and the command `\Q:foo` is undefined, we read everything, and call `\XML@dopi` with innocent arguments.

```
\def\XML@getpi@x#1#2?>{
  \XML@dopi{Undefined}{}}
```

In the case `<?xmltex something?>` we use an auxiliary command that grabs the content with the right category codes. This allows TEX commands, with TEX syntax.

```
\expandafter\def \csname Q:xmltex\endcsname{
  \begingroup
  \XML@reset
  \catcode`\>\active
  \XML@xmltexpi}
```

The code of the the command is trivial. We call `\XML@dopi`.

```
\gdef\XML@xmltexpi#1?>{
  \endgroup
  \XML@dopi{xmltex}{#1}}
```

The default action is trivial also. Not inlined because of grabbing.

```
\def\XML@dopi#1#2{
  #2}
```

2.12 Declarations

In this paragraph, we consider declarations, things that start with `<!`. Here the `\@` has as purpose to read in `#1` all tokens up to the end of `\XML@lt@markup`. We read the first two characters after the `<!` and decide what to do. The `@` at the end makes reading of arguments easy.

```
\def\XML@getdecl#1\@#2#3{
\fi\fi
  \if-\noexpand#2\XML@comment     %   --
  \else\if N\noexpand#3\XML@entity%   EN TITY
  \else\if L\noexpand#3\XML@dec@e%    EL EMENT
  \else\if A\noexpand#2\XML@dec@a%    AT TLIST
  \else\if D\noexpand#2\XML@doctype%  DO CTYPE
  \else\if C\noexpand#3\XML@cdata%    [C DATA
  \else        \XML@dec@n%            NO TATION
@}
```

Easy part: Element declarations are ignored. In fact, elements can only be defined via an xmt file.

```
\def\XML@dec@e#1@#2>{
  \fi\fi\fi
  \XML@checkend@subset}
```

In the case of `<!ATTLIST...>` declarations, we will do something. We start with closing all conditionals. After that, we read the element name and save it somewhere. Then, we parse the list.

```
\def\XML@dec@a#1 #2 {
  \fi\fi\fi\fi
  \protected@xdef\XML@tempa{#2}
  \XML@dec@a@x}
```

The XML production number 52 says that we should have 'AttDef*', optional space and close tag; thus the code fails if no attribute is declared. Let's hope that the list is not empty. Production 53 says that 'AttDef' is space, name, space, 'AttType', space, 'DefaultDecl'. We read the name and store it in `\XML@tempb`. After that we look at the character that follows. It could be an open parenthesis, or something else. The type of the attribute is ignored.

```
\gdef\XML@dec@a@x#1 #2{
  \protected@xdef\XML@tempb{#1}
   \if(\noexpand#2
     \begingroup
     \catcode`\(\active
     \expandafter\XML@dec@a@brack
   \else
      \expandafter\XML@dec@a@type
   \fi}
```

Rule 59 says that the type can be a list enclosed by parentheses.

```
\gdef\XML@dec@a@brack#1){
  \endgroup
  \XML@dec@a@hash}
```

According to rules 54, 55, and 56, the type can be `CDATA`, `ID`, `IDREF`, `IDREFS`, `ENTITY`, `ENTITIES`, `NMTOKEN` or `NMTOKENS`. It could also be `NOTATION` followed by a list. Here we skip over the word, and continue parsing

```
\def\XML@dec@a@type#1 {
  \XML@dec@a@hash}
```

Rule 60 says that we should have `#REQUIRED`, `#IMPLIED`, or a default value, optionally preceded by `#FIXED`. We consider three cases: If we see a `#`, we read it via `\XML@dec@a@type`; this will call this function. Said otherwise, when we are here, we might have read the `#FIXED`, and are ready for the value, or we might have read `#REQUIRED`, and a `>` sign is OK, as well as another attribute declaration; for this reason, we redefine `\ERROR`: this command is called when the character that follows is neither a single quote nor a double quote.

```
\gdef\XML@dec@a@hash$1{
  \if\noexpand$1#
    \expandafter\XML@dec@a@type
  \else
    \ifx$1>
      \let\ERROR\@undefined
      \expandafter\expandafter\expandafter\XML@checkend@subset
    \else
      \let\ERROR\XML@dec@a@nodef
      \XML@dec@a@def$1
    \fi
  \fi}
```

When we come here, we have finished our 'AttDef' and we are ready for the next one.

```
\gdef\XML@dec@a@nodef#1\fi\fi#2{
  \fi\fi
  \XML@dec@a@x#1}
```

When we come here, we have a default value for the attribute.

```
\def\XML@dec@a@def#1\fi\fi{
 \fi\fi
  \XML@quoted\XML@dec@a@default#1}
```

This code adds `\XML@add@attrib{name}{att}{val}` to a global list, where 'name' is the name of the element, 'att' the name of the attribute and 'val' the default value of the attribute. It continues parsing the declaration.

```
\def\XML@dec@a@default#1#2{
  \ifx\XML@default@attributes\relax
    \let\XML@default@attributes\@empty
  \fi
  \toks@\expandafter{\XML@default@attributes}
  \protected@xdef\XML@default@attributes{
    \the\toks@\noexpand\XML@add@attrib{\XML@tempa}{\XML@tempb}{#1}}
  \XML@dec@a@hash#2}
```

Remember line 414: there was `\XML@default@attributes`. This list was constructed by the code above. It consists of a sequence of `\XML@add@attrib ABC`. What we do here is to evaluate in a context where `\begintag` is the element to be evaluated. If it matches, we call `\XML@attribval`. The effect as if the user gave `B='C'`; in the case `B='something'` is on the attribute list, this is evaluated first, and the `B='C'` is useless.

```
\def\XML@add@attrib#1#2#3{
  \gdef\XML@tempa{#1}
  \ifx\XML@tempa\begintag
   \def\XML@this@attribute{#2}
    \let\XML@getattrib\relax
```

```
573     \XML@attribval{#3}
574     \let\XML@getattrib\XML@@getattrib
575   \fi}
```

This reads a comment. The code is trivial. An intermediary command is needed for the case where we want to grab something.

```
576 \def\XML@comment#1@#2-->{
577   \fi
578   \Activespace
579   \XML@comment@}
580 \def\XML@comment@{\XML@checkend@subset}
```

2.13 Entities

We have to distinguish between `<!ENTITY foo ...>` and `<!ENTITY % foo ...>`. The command defined here takes as argument some junk, and what follows, the percent sign or a name. Here in the code, we have two versions of `\XML@input`. This command will be explained later. We continue parsing with `\XML@p@ent` or `\XML@ent`.

```
581 \gdef\XML@entity#1 #2 {
582   \fi\fi
583   \ifx%#2
584   \def\XML@input{
585     \ifx\XML@use\XML@SYSTEM\expandafter\@gobble\else
586       \noexpand\inputonce\fi}
587   \expandafter\XML@p@ent
588   \else
589   \def\XML@input{\noexpand\xmlinput}
590   {\utfeight@protect@chars\xdef\XML@ename{&#2}}
591   \expandafter\XML@ent
592    \fi}
```

We have to distinguish between `<!ENTITY % foo1 "val">`, `<!ENTITY % foo2 SYSTEM "val">`, and `<!ENTITY % foo3 PUBLIC "file" "val">`. Here we put in `\XML@ename` the entity name '`%foo1`' and look at the first character of what follows.

```
593 \gdef\XML@p@ent#1 #2{
594   {\utfeight@protect@chars\xdef\XML@ename{%#1}}
595   \if\noexpand#2P\XML@E@public
596   \else\if\noexpand#2S\XML@E@system
597    \else\XML@E@internal#2}
```

We have to make the same distinctions as in the case `<!ENTITY foo1 ...>`. Here we have put the entity name '`&foo1`' in `\XML@ename` and look at the first character of what follows. This looks like above, but `NDATA` is allowed here.

```
598 \def\XML@ent#1{
599   \if\noexpand#1P\XML@E@public
600   \else\if\noexpand#1S\XML@E@system
601    \else\XML@E@internal#1}
```

This handles the case `<!ENTITY % foo1 "val">` or `<!ENTITY foo2 'val'>`. In `\XML@ename` we have '`%foo1`' or '`&foo2`'. We have read the opening quote. What we do on line 607 is to redefine it to be `</`, so that the parser will see `val</>`. We will read the argument via `\xmlgrab`. This command will be explained later; the important point is that it will read everything up to the end

of the current element (i.e. up to the </>), and call the command associated to the current element, defined on line 608. The effect is to call \XML@E@internal@x after the assignment, which is the \gdef that defines \+%foo1 or \+&foo2 with as body all the grabbed text. The whole difficulty is that the declaration could be something like <!ENTITY ier "<hi rend='sup'>er</hi>">, so that the attribute list has to be parsed, but it is too early for namespace processing. For this reason, some commands have to be redefined.

```
602 \gdef\XML@E@internal#1{
603   \fi\fi
604   \begingroup
605   \let\XML@endgroup\endgroup  % use real groups instead of faked ones.
606   \let\XML@begingroup\begingroup
607   \def#1{</}
608   \expandafter\def\csname E\string/:\endcsname{
609     \afterassignment\XML@E@internal@x
610     \expandafter\gdef\csname+\XML@ename\endcsname}
611   \begingroup
612   \let\XML@ns@decl\relax% stop xmlns 'attribute' being recognised
613   \let\XML@this@local\@empty
614   \def\XML@this@prefix{*} % set up special prefix to gobble colon
615   \let\XML@checkknown\relax % disable these
616   \def\XML@ns##1{% hobble namespace code to put all name in local part.
617     \protected@edef\XML@this@local{##1}
618     \def\XML@this@prefix{*}}
619   \xmlgrab}
```

This closes the group started line 611. After that, we execute three tokens after the current group, namely \XML@trace@warn (for debug), \+&foo2 (argument of previous) and \fihack. The group ends because of the token at line 812 (the \XML@endgroup redefined above).

```
620 \def\XML@E@internal@x{
621    \endgroup
622     \aftergroup\XML@trace@warn
623     \expandafter\aftergroup\csname+\XML@ename\endcsname
624     \aftergroup\fihack
625 }
```

The \fi comes from line 813. This piece of code just ignores the conditional. It continues parsing of the element.

```
626 \def\fihack#1\fi{\expandafter\XML@checkend@subset}
```

When we see <!ENTITY foo PUBLIC 'pub-part' 'system-part'>, this piece of code finishes reading the PUBLIC token, then reads pub-part, and calls \XML@E@pubid.

```
627 \def\XML@E@public#1 {
628    \fi
629    \XML@quoted\XML@E@pubid}
```

After that, all characters in pub-part are converted to category 12, using a classical method, the result is stored in \XML@E@pubid, and system-part is read.

```
630 \def\XML@E@pubid#1{
631   \def\XML@PUBLIC{#1}
632   \edef\XML@PUBLIC{\catxii\XML@PUBLIC}
633   \XML@quoted\XML@E@systemid}
```

The case <!ENTITY foo SYSTEM 'system-part'> is similar, but there is no public part.

```
\def\XML@E@system#1 {
   \fi\fi
   \def\XML@PUBLIC{}
   \XML@quoted\XML@E@systemid}
```

Now we run the catalogue. In the case where 'pub-part' is found as `PUBLIC` item in the catalogue its associated value will be in `\XML@use`. If 'system-part' is found a `SYSTEM` item, the same will be true. Otherwise, `\XML@use` will be `\XML@SYSTEM`. There are two cases to consider. Here `#2` is the first unread character, if it's an N, we have a `NDATA`. Otherwise, we call the function defined below.

```
\def\XML@E@systemid#1#2{
  \def\XML@SYSTEM{#1}
  \let\XML@use\XML@SYSTEM
  \the\XML@catalogue
  \if\noexpand#2N
   \expandafter\XML@E@ndata
  \else
    \afterfi
    \XML@E@internal@{\XML@input{\XML@use}}#2
  \fi}
```

In the case of `NDATA`, we hack bit (this is really strange).

```
\def\XML@E@ndata#1 #2>{\XML@ndata@#2 >}
\def\XML@ndata@#1 #2>{
    \XML@E@internal@{{#1}{\XML@use}}>}
```

The command takes two arguments: some action, and everything that remains on the current element. Assume that we consider `<!ENTITY % foo SYSTEM "bar">`. In this case, we define `\+%foo`; its body will be the expansion of `#1`. This is `\XML@input{\XML@use}`, where the argument is what is found in the catalogue, and the command is defined on lines 584 or 589. In the case of `foo`, without percent sign, the command `\XML@input` is the same as `\xmlinput`. Otherwise, it is `\inputonce` (except: it ignores the argument if not found in the catalogue).

```
\def\XML@E@internal@#1#2>{
   \expandafter\protected@xdef\csname+\XML@ename\endcsname{#1}
  \XML@checkend@subset}}
```

This is done at the end of an `<!ENTITY...>` declaration.

```
\gdef\XML@checkend@subset{
  \Normalspace
  \XML@checkend@subset@}
```

This is a bit strange: why does the command take these four arguments? The definition of the percent character is to make it a normal Unicode character (end of local DTD).

```
\gdef\XML@checkend@subset@#1#2#3#4{
  \ifx]#1
  \let\XML@w@\@empty
  \gdef%{\utfeightay%}
  \let\XML@checkend@subset\relax
  \expandafter\XML@loaddoctype
  \fi
  #1#2#3#4}
```

In the case where the `<!DOCTYPE>` element specifies a DTD, we load the file.

```
\def\XML@loaddoctype#1#2{
  \Activespace
```

```
  \ifx\XML@D@dtd\relax\else
    \inputonce\XML@D@dtd
  \fi}
```

2.14 Interpreting the Doctype element

Consider now that case of `<!DOCTYPE TEI.2 SYSTEM "teilite.dtd">`. This piece of code finishes reading the `DOCTYPE` name, then the name of the document element, it puts it in `\documentelement`. It then checks if what follows is `PUBLIC`, `SYSTEM`, an internal subset, or the end of the element. The command closes all `\fi` that are open. There is an `@` here that makes it easy to skip over the conditionals defined here.

```
\gdef\XML@doctype#1 #2 #3{
 \fi\fi\fi\fi\fi
  \def\documentelement{#2}
  \let\XML@D@dtd\relax
  \if\noexpand#3P\XML@D@public
  \else\if\noexpand#3S\XML@D@system
  \else\ifx#3[\XML@D@internal
  \else%must be > the end
    \XML@D@empty
  @}
```

If nothing is given, we have nothing to do.

```
\gdef\XML@D@empty @{
   \fi\fi\fi}
```

In the case of `PUBLIC`, we parse the public value, and do something with it.

```
\gdef\XML@D@public#1 {
   \fi
   \XML@quoted\XML@pubid}
```

In the case of `<!DOCTYPE foo PUBLIC "aaa" "bbb">`, we put the `aaa` part in `\XML@PUBLIC`, change all category codes to 12, and read the '`bbb`' part.

```
\gdef\XML@pubid#1{
  \def\XML@PUBLIC{#1}
  \edef\XML@PUBLIC{\catxii\XML@PUBLIC}
  \XML@quoted\XML@systemid}
```

In the case of `<!DOCTYPE foo SYSTEM "bbb">`, it is as above, but the public part is empty.

```
\gdef\XML@D@system#1 {
   \fi\fi
   \def\XML@PUBLIC{}
   \XML@quoted\XML@systemid}
```

We put the `bbb` part in `\XML@SYSTEM`, change all category codes to 12, run the catalogue, and put the result in `\XML@D@dtd` for later use by `\XML@loaddoctype`.

```
\gdef\XML@systemid#1{
  \protected@edef\XML@SYSTEM{#1}
  \edef\XML@SYSTEM{\catxii\XML@SYSTEM}
  \let\XML@use\@empty
  \the\XML@catalogue
  \let\XML@D@dtd\XML@use
  \XML@D@internal@}
```

When we have no `PUBLIC` and no `SYSTEM` part, but only a local DTD, we call this: it pops the conditional stack, and pushes back the open bracket. The action is the same as if a `PUBLIC` or `SYSTEM` part had be given.

```
\gdef\XML@D@internal#1@{
  \fi\fi\fi
  \XML@D@internal@[}
```

We parse the internal DTD in the same fashion as everything else. However `%foo;` evaluates to something, so that the percent sign must be activated. The vertical bar is used for comments.

```
\gdef\XML@D@internal@#1{
  \ifx[#1
    \let%\XML@pcent
    \edef\XML@w@{ \XML@w@}
     \expandafter\XML@checkend@subset
  \else
      | it had better be the closing >
   \fi}
```

Inside a local DTD, the parameter entity '`%foo;`' evaluates to `\+%foo`.

```
\gdef\XML@pcent#1;{
  \csname+%#1\endcsname
  \XML@checkend@subset}
```

When you say `&#foo;` the `\XML@charref` command is called to parse the entity; the result is in `\XML@tempa`. It will in general be evaluated right now. On the other hand '`&foo;`' evaluates to `\+&foo`, there is no intermediate command.

```
\let&\XML@amp@markup
\gdef\XML@amp@markup$1$2;{
  \ifx#$1\@empty
   \XML@charref$2;
   \XML@tempa
  \else
   \begingroup\utfeight@protect@chars
   \expandafter\aftergroup
   \csname+\string&$1$2\expandafter\endcsname
   \endgroup
  \fi}
```

In the case of `<![CDATA xxx ]]>`, this reads up to the first space.

```
\gdef\XML@cdata #1[{
 \fi\fi\fi\fi\fi\fi
  \Activespace
  \XML@cdata@a}
```

And this reads everything up to the special end marker `]]>`. Less than sign and ampersand are not active.

```
\gdef\XML@cdata@a#1]]>{
  \begingroup
  \edef<{\noexpand\utfeightaz\string<}
  \edef&{\noexpand\utfeightaz\string&}
  \XML@docdata{#1}}
```

The action here is trivial. We need an intermediary command, in the case of grab.

```
\def\XML@docdata#1{#1\endgroup}
```

The only thing done here is to skip over everything, until the end.

```
\def\XML@dec@n#1N #2 #3 {
 \fi\fi\fi\fi\fi\fi
  \XML@quoted\XML@notation
  }

\def\XML@notation#1#2{
  \ifx>#2
   \expandafter\XML@checkend@subset
  \else
    \afterfi
    \XML@quoted\XML@notation#2
  \fi}
```

2.15 Grabbing content

The normal behavior of `<foo>text</foo>` is like `\begin{foo}text\end{foo}`. This method is the most efficient concerning memory space. In some cases, we prefer the equivalent of `\def\arg{text}`, `\foo{\arg}`. The interesting point is that, assuming that `<foo>` takes two children, and that we do not care about cases with incorrect syntax, we can manage everything so that the user function sees these two children, for instance in the form `\split\arg\first\second` followed by `\foo\first\second`. In fact, instead of calling a single command, we call two commands, as `\foofirst\first`, `\foosecond\second`.

The following piece of code is the definition of the `<msup>` element in the MathML namespace. We shall explain the syntax of `\XMLelement` later. Line 10002 says: we do not care about attributes, line 10003 says that we want to grab the content of the element. Line 10004 says: there are two children, and we want to apply some commands to them.

```
10000   \XMLelement{m:msup}
10001     {}
10002     {\xmlgrab}
10003     {\xmltextwochildren\@firstofone\sp#1}
```

We shall define `\xmlgrab` below. It will read the content of the element, and use `\@empty` as element separator (see code line 819); remember that `\@empty` expands to nothing, hence is harmless. This marker allows easy splitting. In the case of the example of the start of the chapter, the tokens are

```
\xmltextwochildren\@firstofone\sp
+<3:mi^^I>L</3:mi>\@empty  <3:mn^^I>2</3:mn>\@empty  +
```

The second line is printed by TEX, when we ask for the value of `+#1+`, we has inserted the plus signs, this being the easiest way to see that each `\@empty` is followed by a space (they are in the input file), and for each opening tag, a tabulation between the tag name and the attribute list (which is empty in this case). With the definition below, the arguments of `\xmltextwochildren` will be

```
#1=\@firstofone
#2=\sp
#3=<3:mi>L</3:mi>
#4=<3:mn>2</3:mn>
```

(we did not show the tabulations, nor the spaces). Tabulations are read again by the parser when looking for attributes, spaces are ignored, as usual, in math mode. The effect of the command is to apply the first argument to the third, the second to the fourth. If the arguments are, say, `\A`,

`\B`, CC and DD, the result is `\A{CC}\B{DD}`. In our case, we want `CC^{DD}`, so that `\A` is just a command that removes useless braces, and `\B` is the TeX primitive for superscripts.

```
\def\xmltextwochildren#1#2#3\@empty#4\@empty{
  #1{#3}#2{#4}}
\def\xmltexthreechildren#1#2#3#4\@empty#5\@empty#6\@empty{
  #1{#4}#2{#5}#3{#6}}
```

This is a small function that returns everything before the `\@empty`. You have to use `\@` to mark the end of the child list (only first child is used here).

```
\def\xmltexfirstchild#1\@empty#2\@{
  #1}
```

If you say `\xmltexforall\cmd{list}`, where the second argument is a list of tokens with `\@empty` between tokens, this applies `\cmd` to each item. Moreover, the quantity `\xml@name` contains the name of the element. The end of the loop relies on the fact that no element name starts with a space.

```
\def\xmltexforall#1#2{
  \xmltexf@rall#1#2< >\@empty}

\def\xmltexf@rall#1#2<#3 #4>#5\@empty{
  \ifx\relax#3\relax
  \else
  \def\xml@name{#3}#1{<#3 #4>#5}
  \expandafter\xmltexf@rall\expandafter#1
  \fi}
```

The action associated to `<foo>` consists in two parts: first all attributes are scanned, and some commands are instantiated (see section 2.10), and then the start code is executed (in the example, line 10002). When we see `</foo>`, we execute the end code (line 10003, in the example). In the special case where the initial action is `\xmlgrab`, the command gets an argument. This argument is computed on line 811 as the value of the token list `\XMLgrabtoks`. Thus, the `\xmlgrab` command must read all tokens, up to the end tag; it must handle namespaces properly (as the example shows, all namespaces, even the default ones, are replaced by integers).

The idea is to redefine temporarily all commands `\XML@do...`, for the case `<foo>`, `<?foo>`, `<!foo>`, and `</foo>`, and ask them to put the result in the list. We store in `\XML@next@level` the value of `\XML@w@` at the next level. It is thus possible to check, for a given element, if it is a child (and not merely a descendant) of the current element, so that we know where to insert the `\@empty` markers. The main routine here is `\grab@`.

```
\def\xmlgrab{
  \begingroup
  \global\XMLgrabtoks{}
  \let\XML@this@level\XML@w@
  \edef\XML@next@level{ \XML@w@}
  \let\XML@doelement\XML@grabelement
  \let\XML@doend\XML@grabend
  \let\XML@docdata\XML@grabcdata
  \let\XML@comment@\XML@grabcomment@
  \let\XML@dopi\XML@grabpi
  \XMLgrab@}
```

This uses the same magic as `\XML@getname`. The idea is to read everything until the next less-than sign, putting all tokens in the command `\XML@tempa`.

```
\def\XMLgrab@{
  \utfeight@protect@internal
  \def<{\iffalse{\fi}\XMLgrab@@}
  \xdef\XML@tempa{\iffalse}\fi}
```

When `\XMLgrab@` has read everything between tags, it puts the grabbed tokens in the token register `\XMLgrabtoks`, and then evaluates the less-than sign.

```
\def\XMLgrab@@{
  \global\XMLgrabtoks\expandafter{\the\expandafter\XMLgrabtoks\XML@tempa}
  \XML@lt@markup}
```

This command is called when we grab the content of an element. Assume that we have seen `<mi>L</mi>`. When we are here, we have seen the first less-than sign. And we know that `\XML@this@element` is '`3:mi`'. We add to the token list `<3:mi atts>`. There is a tabulation after the element name, this is obtained by uppercasing the tilde. Attributes are added by a call to `\the\XML@attribute@toks`, with a temporary redefinition of `\XML@doattribute`, and the default namespace is neutralized.

```
\uppercase{
\gdef\XML@grabelement{
   \Activespace
  \global\XMLgrabtoks\expandafter{
    \the\expandafter\XMLgrabtoks
      \expandafter<\XML@this@element~}
   \begingroup
   \let\XML@doattribute\XML@grabattribute
   \def\XMLNS@{0}
   \expandafter\let\csname XMLNS@0\endcsname\XMLNS@
   \the\XML@attribute@toks
   \endgroup
   \Activespace
  \global\XMLgrabtoks\expandafter{
    \the\XMLgrabtoks
    >}
   \XMLgrab@}
}
```

Assume that `foo:bar='gee'` is in the attribute list of the current element, and assume that `foo` has namespace number 4. We add `4:bar="gee"` and a space to the token list.

```
\gdef\XML@grabattribute#1#2#3{
  \protected@xdef\XML@tempa{\jg@namespace{#1}:#2}
  \global\XMLgrabtoks\expandafter{
    \the\expandafter\XMLgrabtoks
    \XML@tempa="#3" }}
```

Let's assuming that we are grabbing something and we see `</foo>`. There are two cases to consider. If this element is the one we are looking for, we close the group open by `\xmlgrab`, and we execute the command `\E/:3:msup` (there are some hacks here; the `\uppercase` command replaces dot and star by slash and colon with the right category code). We pass the grabbed token list as argument. This is achieved by putting an `\expandafter` just before the `\endcsname`, it will expand `\the`, i.e., replace `\XMLgrabtoks` by its value (since we want braces around this token list, another `\expandafter` is needed). After execution of the command, we have to close our XML group and hack with category codes. On the other hand, in the case where the element does not end grabbing, we add `</4:foo>` to the end of the `\XMLgrabtoks` token list. If this is a direct child, we add also a `\@empty` marker. We continue grabbing via a call to `\XMLgrab@`.

```
\uppercase{
\gdef\XML@grabend{
  \ifx\XML@this@level\XML@w@
    \endgroup
    \csname
      E.*\jg@this@namespace
        *\XML@this@local
    \expandafter\endcsname\expandafter{
      \the\XMLgrabtoks}
    \XML@endgroup
    \ifnum\catcode`\^^M=10  \Activespace \fi
  \else
    \xdef\XML@tempa{\noexpand<\noexpand/
      \jg@this@namespace\noexpand: %%%% \expandafter omitted [jg]
            \XML@this@local
    \noexpand>
    \ifx\XML@next@level\XML@w@\noexpand\@empty\fi}
    \global\XMLgrabtoks\expandafter{
      \the\expandafter\XMLgrabtoks
      \XML@tempa}
    \XML@endgroup
  \expandafter
    \XMLgrab@
  \fi}}
```

When we want to grab something and see `<?PI etc?>`, we add all these tokens to our list.

```
\gdef\XML@grabpi#1#2{
  \global\XMLgrabtoks\expandafter{
  \the\XMLgrabtoks<?#1^^I#2?>}
  \XMLgrab@}
```

If you say `\NDATAEntity\att\A\B`, if the expansion of `\att` is something like `foo`, and `\&+foo` expands to `\bar` and `\gee`, this piece of code applies `\A` to `\bar` and `\B` to `\gee`[9].

```
\gdef\NDATAEntity#1{
  \expandafter\expandafter\expandafter
  \XML@ndataentity\csname+&#1\endcsname}

\gdef\XML@ndataentity#1#2#3#4{
  #3{#1}#4{#2}}
```

Grabbing `CDATA` is easy: what we do is re-insert the content of the element, and continue grabbing. Ampersands and less-than signs are replaced by the equivalent of `&` and `<`, said otherwise, inactive characters.

```
\def\XML@grabcdata#1{
  \utfeight@protect@internal
  \edef<{\noexpand\utfeightaz\string<}
  \edef&{\noexpand\utfeightaz\string&}
  \xdef\XML@tempa{#1}
  \endgroup
   \expandafter\XMLgrab@\XML@tempa}
```

When we grab a comment, the only thing we need to do is continue grabbing.

[9] This is unclear to me

```
\def\XML@grabcomment@{
  \XMLgrab@}
```

2.16 Defining actions

This is for use in a .xmt file, the file that defines actions for each element. After `\XMLentity{foo}{bar}`, the XML entity `&foo;` evaluates to `bar`.

```
\gdef\XMLentity#1#2{
  \expandafter\gdef\csname+&#1\endcsname{#2}}
```

These are the predefined entities:

```
\XMLentity{amp}{\utfeightaz&}
\XMLentity{quot}{\utfeightax"}
\XMLentity{apos}{\utfeightax'}
\XMLentity{lt}{\utfeightaz<}
\XMLentity{gt}{\utfeightax>}
```

The `\XMLelement` command appears in a .xmt file. It takes four arguments: the first one is the name of an element. We have seen an example above. Assume that the name is `m:msup`. Assume that the 'm' prefix stands for MathML and this corresponds to the number 3. We define two commands `\E:3:msup` and `\E/:3:msup`. The body of these commands is argument `#3` and `#4`. In the case where `#3` is `\xmlgrab` then the `\E/:3:msup` command takes an argument (see previous section). Argument `#2` explains what to do with attributes. It should contain a sequence of `\XMLattribute` commands. Note: all attributes declared for the current namespace (found in `\A:3`) are added to the list. A call to evaluation of the attribute list is inserted in the body of `\E:3:msup` before the user code. On line 860, 861, the purpose of all these `\expandafter` is to expand the `\the`, so as to insert the token list (and not a reference to it) in the body of the definition.

```
\long\def\XMLelement#1#2#3#4{
  \XML@ns{#1}
  \xdef\XML@tempc{:\jg@this@namespace
       :\XML@this@local}
  \toks@\expandafter{\csname A:\jg@this@namespace
    \endcsname}
  #2
  \expandafter\gdef\csname E\XML@tempc\expandafter\endcsname
  \expandafter{\expandafter\XML@setattributes\expandafter{\the\toks@}#3}
  \gdef\XML@tempa{#3}
  \ifx\XML@tempa\XML@xmlgrab
    \expandafter\gdef\csname E\string/\XML@tempc\endcsname##1
    {#4}
  \else
    \expandafter\gdef\csname E\string/\XML@tempc\endcsname
    {#4}
  \fi}
\def\XML@xmlgrab{\xmlgrab}
```

When you say `\XMLattribute{form}{\mycmd}{inline}` this puts in `\XML@tempa` the quantity `\XML@attrib 0:form \relax \mycmd {inline}`. If '`form`' is replaced by '`m:form`' and the namespace value of '`m`' is 3, then '`3:form`' will be used instead of '`0:form`'. This works by redefining locally the default namespace to be the empty namespace The value of `\XML@tempa` will be added to the end of `\toks@`, the token list used in `\XMLelement` or `\XMLnamespaceattribute`. For some strange

reason the second argument is put in `\XML@tempa`, but not the last one (this implies that the third argument is not expanded; the single token of the second argument is not expanded, because of the `\noexpand`; if the second argument has more than one token, you lose).

```
\long\def\XMLattribute#1#2#3{
  {\def\XMLNS@{0}
  \XML@ns{#1}
  \xdef\XML@tempa{\noexpand\XML@attrib
      \jg@this@namespace
        :\XML@this@local\relax\noexpand#2}}
  \toks@\expandafter{\the\expandafter\toks@\XML@tempa{#3}}}
```

Like above, with a little hack. Assume that the attribute has to be stored in `\foo`. Then `\utfeight@chardef\foo` is executed (some time after a value has been stored).

```
\long\def\XMLattributeX#1#2#3{
  {\def\XMLNS@{0}
  \XML@ns{#1}
  \xdef\XML@tempa{\noexpand\XML@attrib
      \jg@this@namespace
        :\XML@this@local\relax\noexpand#2}}
  \toks@\expandafter{\the\expandafter\toks@\XML@tempa{#3}\utfeight@chardef#2}}
```

This is the action associated to the special setting used above. The command is fully expanded, in a group where this is harmless, globally put in a temporary, and then, outside the group, the temporary is put again in the command. There seems to be a problem here: what if the argument contains ampersands and less-than signs?

```
\def\utfeight@chardef#1{
\begingroup
\utfeight@protect@chars
\xdef\x@temp{#1}
\endgroup
\let#1\x@temp}
```

In case `\XMLnamespaceattribute{foo}{bar}{gee}{etc}`, if the namespace number of `foo` is 4, this code will define the action associated to the attribute defined by bar, gee, etc, and put it in `\A:4`. This command should be used at toplevel, not inside `\XMLelement`.

```
\long\def\XMLnamespaceattribute#1#2#3#4{
   \toks@\expandafter\expandafter\expandafter{\csname A:%
    \jg@namespace{#1}\endcsname}
  \XMLattribute{#2}{#3}{#4}
  \expandafter\xdef\csname A:\jg@namespace{#1}\endcsname{\the\toks@}}
```

Idem, expanded.

```
\long\def\XMLnamespaceattributeX#1#2#3#4{
   \toks@\expandafter\expandafter\expandafter{\csname A:%
    \jg@namespace{#1}\endcsname}
  \XMLattributeX{#2}{#3}{#4}
  \expandafter\xdef\csname A:\jg@namespace{#1}\endcsname{\the\toks@}}
```

If you say `\XMLname{foo:bar}{\gee}` this puts in `\gee` something like `4:bar`.

```
\long\gdef\XMLname#1#2{{
  \XML@ns{#1}
  \xdef#2{\jg@this@namespace\noexpand:\XML@this@local}}}
```

If you say `\XMLstring\foo<>bar</>` this piece of code reads the `<>`, then calls `\xmlgrab`, which reads everything until the `</>`, and executes the code associated to the end tag: this defines `\foo`, and closes the group.

```
\gdef\XMLstring#1#2<>{
  \begingroup
  \let\XML@endgroup\endgroup
  \let\XML@this@local\@empty
  \let\XML@this@prefix\@empty
  \expandafter\def\csname E/:\XMLNS@:\endcsname{\gdef#1}
  \XML@catcodes
  \xmlgrab}
```

Same code, expanded.

```
\gdef\XMLstringX#1#2<>{
  \begingroup
  \let\XML@endgroup\endgroup
  \let\XML@this@local\@empty
  \let\XML@this@prefix\@empty
  \expandafter\def\csname E/:\XMLNS@:\endcsname{\xdef#1}
  \XML@catcodes
  \utfeight@protect@chars
  \xmlgrab}
```

2.17 Other commands

Public version of `\XML@setenc`.

```
\let\DeclareNamespace\XML@ns@uri  %  version for xmt files
\def\FileEncoding#1{\XML@setenc{#1}\relax}
\newtoks\XML@attribute@toks
\newcount\XML@ns@count
\newtoks\XMLgrabtoks
```

The next function reads an XML file. The idea is to restore the current encoding. We have not shown the source of `\XML@xmlinput`.

```
\def\xmlinput#1{
 \IfFileExists{#1}
  {\expandafter\XML@xmlinput\expandafter
    \XML@setenc\expandafter{\XML@thisencoding}\relax
  }{\XML@warn{No file: #1}}}

\def\XML@xmlinput{...}
```

This reads a TEX file only once.

```
\def\inputonce#1{
  \expandafter\ifx\csname xmt:#1\endcsname\relax
  \global\expandafter\let\csname xmt:#1\endcsname\@ne
  \begingroup
  \XML@reset
  \def\XMLNS@{0}
  \input{#1}
  \endgroup
  \fi}
```

In case you want some characters to be active.

```
\gdef\ActivateASCII#1{
  \uppercase{\count@"0\if x\noexpand#1\relax\else\count@#1\fi\relax}
  \toks@\expandafter{\nfss@catcodes}
       \xdef\nfss@catcodes{
       \catcode\the\count@=\the\catcode\the\count@\relax\the\toks@}
  \toks@\expandafter{\XML@catcodes}
     \xdef\XML@catcodes{
       \catcode\the\count@\active\the\toks@}
  \expandafter\ifx\csname8:"\endcsname\relax
    \expandafter\gdef\csname8:"\endcsname{"}
  \fi}
\ActivateASCII{94}% ^ for tex ^^ notation in aux files
\ActivateASCII{x5C}% \
\ActivateASCII{x5F}%  underscore [jg]
\ActivateASCII{123}% {
\ActivateASCII{125}%  close brace [jg]
```

We redefine `<obeyspaces>` and `<obeylines>`. The idea is to redefine the action associated to space and newline character.

```
\expandafter\def\expandafter\obeylines\expandafter{
\expandafter\def\csname 8:\string^^M\endcsname{\leavevmode\hfil \break\null}}

\expandafter\def\expandafter\obeyspaces\expandafter{
\expandafter\def\csname 8: \endcsname{\nobreakspace}}
```

This line is a bit strange:

```
\expandafter\def\csname 8:\string^^I\expandafter\endcsname
       \expandafter{\csname 8: \endcsname}
```

The end of the xmltex.tex file is like this (we simplified a bit the code, by assuming that we do not want to dump a format, and that `\xmlfile` is defined). Essentially, we reset the end-of-line character to it normal meaning, we set the category codes, and we load the XML file.

```
\def\XML@tempa{\catcode`\-12\relax\input\xmlfile\relax}
\endlinechar`\^^M \expandafter\XML@catcodes\XML@tempa
```

Chapter 3

Interpreting MathML and related stuff in TeX

In the previous chapter, we have seen that TeX is able to read and evaluate an XML file. The Raweb is not typeset directly: there is a stylesheet that converts it into XSL/Format, see chapter 7. The purpose of the next chapter is to explain how XSL/Format has to be interpreted by TeX; in this chapter we explain everything else, the big part being MathML, described in [2].

This file was originally named mathml2.xmt, it is now called raweb-cfg.sty. It will be loaded in case your file is called foo.tex (more precisely if `\jobname` contains 'foo') and this file is renamed foo.cfg. This file is read before the XML file is loaded (in particular, before the file described in the next chapter). It starts like this

```
% patch of xmltex.tex
%% Copyright 2000 David Carlisle, for the original mathml part
%% Copyright 2004 Jose Grimm

%% This file is distributed under the LaTeX Project Public License
%% (LPPL) as found at http://www.latex-project.org/lppl.txt
%% Either version 1.0, or at your option, any later version.
```

3.1 Local patches to xmltex.tex

Redefinition of `\utfeight@protect@chars`. Compare this with the definition on page 12. The idea is that using `\let` rather than `\def` should be more efficient. Some tests show that the speedup factor is nearly 10 percent.

```
\def\utfeight@protect@chars{%
  \let\utfeightax\string
  \let\utfeightay\string
  \let\utfeightaz\string
  \let\utfeightb\utfeightb@jg@
  \let\utfeightc\utfeightc@jg@
  \let\utfeightd\utfeightd@jg@}
```

Three auxiliary definitions used above.

```
\def\utfeightb@jg@#1#2{#1\string#2}
\def\utfeightc@jg@#1#2#3{#1\string#2\string#3}
\def\utfeightd@jg@#1#2#3#4{#1\string#2\string#3\string#4}
```

We redefine this also (original definition on page 11).

```
\begingroup
\catcode‘<\active
\catcode‘&\active
\gdef\utfeight@protect@internal{%
  \let\utfeightax\noexpand
  \let\utfeightay\noexpand
  \let\utfeightaz\utfeightaz@jg@int
  \let<\relax\let&\relax
  \let\utfeightb\utfeightb@jg@int
  \let\utfeightc\utfeightc@jg@int
  \let\utfeightd\utfeightd@jg@int}
```

These are the auxiliary commands.

```
\def\utfeightaz@jg@int{\noexpand\utfeightaz\noexpand}
\def\utfeightb@jg@int#1#2{\noexpand\utfeightb#1\string#2}
\def\utfeightc@jg@int#1#2#3{\noexpand\utfeightc#1\string#2\string#3}
\def\utfeightd@jg@int#1#2#3#4{\noexpand\utfeightd#1\string#2\string#3\string#4}
```

This is redefined (original on page 39) so that we can use our definition of `\XML@amp@markup` shown below.

```
\gdef\utfeight@chardef#1{%
  \begingroup
  \utfeight@protect@chars
  \let&\XML@amp@markup@jg      % <-  modified
  \xdef\x@temp{#1}%
  \endgroup
  \let#1\x@temp}
\endgroup
```

The fotex.xmt file contains a declaration of the form
‘`\XMLnamespaceattributeX {fo} {external-destination} {\FOexternaldestination} {}`’
The ‘X’ after the command name means that the argument is evaluated in an `\xdef`. In the case where we want a `&` in an URL, this does not work. Thus we redefine the meaning of `&`: in the case of `&` the expansion is `&`, otherwise the standard behavior is used.

```
\def\XML@amp@markup@jg#1;{%
  \XML@amp@markup@jgw#1;amp;\@nil}
\def\XML@amp@markup@jgw#1amp;#2\@nil{%
  \ifx b#2b\XML@amp@markup#1;\else\string&\fi}
```

3.2 Support for MathML

We pretend here that mathml2.xmt has been loaded.

```
\global\expandafter\let\csname xmt:mathml2.xmt\endcsname\@ne
```

We declare the namespaces.

```
\DeclareNamespace{m}{http://www.w3.org/1998/Math/MathML}
\DeclareNamespace{fotex}{http://www.tug.org/fotex}
\DeclareNamespace{fo}{http://www.w3.org/1999/XSL/Format}
```

Let's start with simple things: `<m:mrow>`*text*`</m:mrow>` translates to `\bgroup` *text* `\egroup`. This differs from the original coding, allowing constructions like `${x^2}^3$`.

```
\XMLelement{m:mrow}
  {}
  {\bgroup}
  {\egroup}
```

Processing of `<m:msub>` *base subscript* `</m:msub>`: the translation is `base_{subscript}`. The MathML standard says that there is a subscriptshift attribute that specifies the minimum amount to shift the baseline of subscript down. This is not implemented.

```
\XMLelement{m:msub}
  {}
  {\xmlgrab}
  {\xmltextwochildren\@firstofone\sb#1}
```

Processing of `<m:msup>` *base supscript* `</m:msup>`: the translation is `base^{supscript}`. The attribute superscriptshift defined by the standard is ignored.

```
\XMLelement{m:msup}
  {}
  {\xmlgrab}
  {\xmltextwochildren\@firstofone\sp#1}
```

Processing of `<m:msubsup>` *base subscript superscript* `</m:msubsup>`: the translation is `base_{subscript}^{superscript}`. The attributes subscriptshift and superscriptshift defined by the standard are ignored.

```
\XMLelement{m:msubsup}
  {}
  {\xmlgrab}
  {\xmltexthreechildren\@firstofone\sb\sp#1}
```

Processing of `<m:mstyle` *atts*`>` *text* `</m:mstyle>`. The translation is *text*. However, the attributes of the element influence typesetting of it. The attributes veryverythinmathspace, verythinmathspace, thinmathspace, mediummathspace, thickmathspace, verythickmathspace, and veryverythickmathspace can be modified (default value k/18em, for k=1 to 7). TEX provides only three values: thin, medium and thick. The attribute scriptsizemultiplier gives the quotient of the font size at level N and $N+1$ and scriptminsize indicates the minimum size (TEX has three levels of math fonts: text, script, and scriptscript; there is no such limit in MathML, however there is a limit on the size). The default value of the multiplier is 0.71. If the main font is a 10pt, then the script size will be 7pt, and scriptscript size will be 5pt (these are the default values or TEX; however the default scriptminsize is 8pt). We ignore the background attribute.

In this piece of code, we consider the value of the displaystyle attribute. If it is true, we evaluate `\displaystyle`. If it is neither true nor false, nothing happens. If it is false, we look at the scriptlevel attribute. We execute `\textstyle`, or `\scriptstyle`, or `\scriptscriptstyle`, if it is 0, 1 or 2. If the attribute is not provided, we use 0 as default value; if the value is out of range, we use 2.

```
\XMLelement{m:mstyle}
 {\XMLattribute{displaystyle}{\XML@att@displaystyle}{foo} %
   \XMLattribute{scriptlevel}{\XML@scriptlevel}{0}%
 }
 {\xmlgrab}
 {{\ifx\XML@att@displaystyle\att@true\displaystyle
   \else\ifx\XML@att@displaystyle\att@false
    \ifx\XML@scriptlevel\att@dzero\textstyle
```

```
73      \else\ifx\XML@scriptlevel\att@done\scriptstyle
74       \else \scriptscriptstyle\fi\fi
75    \fi %do nothing if neither true nor false
76    \fi#1}}
```

Processing of `<m:mroot>` ***base index*** `</m:mroot>`. We call `\mathmlroot` with ***base*** and ***index*** as arguments (note that the empty group `{}` disappears when this calls the intermediate command). The MathML norm says that the scriptlevel of ***index*** should be increased by two (thus, it will be as small as the TEX version).

```
77 \XMLelement{m:mroot}
78   {}
79   {\xmlgrab}
80   {\xmltextwochildren\mathmlroot{}#1}
81
82 \def\mathmlroot#1#2{\root#2\of{#1}}
```

Processing of `<m:msqrt>` ***base*** `</m:msqrt>`. We just call `\sqrt`.

```
83 \XMLelement{m:msqrt}
84   {}
85   {\xmlgrab}
86   {\sqrt{#1}}
```

Processing of `<m:mtext>...</m:mtext>`. That's easy, we use `\text`.

```
87 \XMLelement{m:mtext}
88   {}
89   {\xmlgrab}
90   {\text{#1}}
```

Processing of `<m:ms>...</m:ms>`. Not yet implemented. The MathML doc says: The `<m:ms>` element is used to represent "string literals" in expressions meant to be interpreted by computer algebra systems.

```
91 % \XMLelement{m:ms} {} {"} {"}
```

Processing of `<m:mn>...</m:mn>`. The MathML doc says: Unlike `<mi>`, `<mn>` elements are (typically) rendered in an unslanted font by default, regardless of their content.

```
92 \XMLelement{m:mn}
93   {}
94   {\xmlgrab}
95   {\mathrm{#1}}
```

Processing of `<m:mi mathvariant=`***xx***`>...</m:mi>`. The MathML doc says that the fontstyle attribute is deprecated, and the mathvariant attribute can take one of the following values: normal, bold, italic, bold-italic, double-struck, bold-fraktur, script, bold-script, fraktur, sans-serif, bold-sans-serif, sans-serif-italic, sans-serif-bold-italic, monospace.

```
96 \XMLelement{m:mi}
97   {\XMLattribute{mathvariant}{\XML@mathmlvariant}{normal}}
98   {\xmlgrab}
99   {\mi@test#1\relax}
```

As you can see, not all variants are implemented here. On the other hand, the test done on line 107 is false in the case where the content of the element is a letter plus some space. (The MathML norm does not says how to get a single letter using an upright variant).

```
100 \gdef\mi@test#1#2\relax{
101  \ifx\XML@mathmlvariant\att@mathml@bold
102   \mathbf{#1#2}\else
```

```
\ifx\XML@mathmlvariant\att@mathml@sansserif
 \mathsf{#1#2}\else
\ifx\XML@mathmlvariant\att@mathml@tt
 \texttt{#1#2}\else
 \ifx\mi@test#2\mi@test
  \expandafter#1
 \else
  \mathrm{#1#2}
 \fi\fi\fi\fi}
```

These are used in the code above.

```
\XMLstring\att@mathml@bold<>bold</>
\XMLstring\att@mathml@sansserif<>sans-serif</>
\XMLstring\att@mathml@tt<>monospace</>
```

Processing of `<m:mspace atts></m:mspace>`. The MathML doc says that attributes width, height, depth, linebreak are allowed. Here we consider only the width. The `\@defaultunits` command is called for the case where a dimension is given without a unit.

```
\XMLelement{m:mspace}
  {\XMLattribute{width}{\XML@mspacewidth}{0}}
  {}
  {\@defaultunits\dimen@\XML@mspacewidth pt\relax\@nnil
   \ifnum\dimen@=\z@\else\kern\dimen@\fi}
```

3.2.1 Fences

Processing of `<m:mfenced open=A close=B separators=...>content</m:mfenced>`. We ignore the separators. The translation is `\leftA content\rightB`. However, what `\left` and `\right` want are numbers (to be precise, slots into math fonts that indicate how large delimiters can be made from smaller pieces); what TeX wants is a character with a `\delcode`, what the XML file contains is a character. We hack a bit, because we cannot expand all these Unicode characters.

```
\XMLelement{m:mfenced}
  { \XMLattribute{open}{\XML@fenceopen}{(}
    \XMLattribute{close}{\XML@fenceclose}{)}
  }
  {\jg@hacko\left\XML@fenceopen}
  {\jg@hackc\right\XML@fenceclose}
```

This code is easy: for instance, if the character is U+2329, we replace it by `\langle`.

```
\def\jg@hacko{%
  \ifx\XML@fenceopen\jg@lt\let\XML@fenceopen\langle\fi
  \ifx\XML@fenceopen\jg@lbra\let\XML@fenceopen\langle\fi
  \ifx\XML@fenceopen\jg@xlbra\let\XML@fenceopen\langle\fi
  \ifx\XML@fenceopen\jg@gt\let\XML@fenceopen\rangle\fi
  \ifx\XML@fenceopen\jg@rbra\let\XML@fenceopen\rangle\fi
  \ifx\XML@fenceopen\jg@xrbra\let\XML@fenceopen\rangle\fi
  \ifx\XML@fenceopen\jg@verbar\let\XML@fenceopen\|\fi
  \ifx\XML@fenceopen\jg@Verbar\let\XML@fenceopen\|\fi
  \ifx\XML@fenceopen\jg@lfloor\let\XML@fenceopen\lfloor\fi
  \ifx\XML@fenceopen\jg@rfloor\let\XML@fenceopen\rfloor\fi
  \ifx\XML@fenceopen\jg@lceil\let\XML@fenceopen\lceil\fi
  \ifx\XML@fenceopen\jg@rceil\let\XML@fenceopen\rceil\fi
```

```
  \ifx\XML@fenceopen\jg@lmoustache\let\XML@fenceopen\lmoustache\fi
  \ifx\XML@fenceopen\jg@rmoustache\let\XML@fenceopen\rmoustache\fi
  \ifx\XML@fenceopen\jg@lgroup\let\XML@fenceopen\lgroup\fi
  \ifx\XML@fenceopen\jg@rgroup\let\XML@fenceopen\rgroup\fi
  \ifx\XML@fenceopen\jg@uparrow\let\XML@fenceopen\uparrow\fi
  \ifx\XML@fenceopen\jg@downarrow\let\XML@fenceopen\downarrow\fi
  \ifx\XML@fenceopen\jg@Uparrow\let\XML@fenceopen\Uparrow\fi
  \ifx\XML@fenceopen\jg@Downarrow\let\XML@fenceopen\Downarrow\fi
  \ifx\XML@fenceopen\jg@updownarrow\let\XML@fenceopen\updownarrow\fi
  \ifx\XML@fenceopen\jg@Updownarrow\let\XML@fenceopen\Updownarrow\fi
}
```

Same code for closing delimiters.

```
\def\jg@hackc{%
  \ifx\XML@fenceclose\jg@gt\let\XML@fenceclose\rangle\fi
  \ifx\XML@fenceclose\jg@rbra\let\XML@fenceclose\rangle\fi
  \ifx\XML@fenceclose\jg@xrbra\let\XML@fenceclose\rangle\fi
  \ifx\XML@fenceclose\jg@lt\let\XML@fenceclose\langle\fi
  \ifx\XML@fenceclose\jg@lbra\let\XML@fenceclose\langle\fi
  \ifx\XML@fenceclose\jg@xlbra\let\XML@fenceclose\langle\fi
  \ifx\XML@fenceclose\jg@verbar\let\XML@fenceclose\|\fi
  \ifx\XML@fenceclose\jg@Verbar\let\XML@fenceclose\|\fi
  \ifx\XML@fenceclose\jg@lfloor\let\XML@fenceclose\lfloor\fi
  \ifx\XML@fenceclose\jg@rfloor\let\XML@fenceclose\rfloor\fi
  \ifx\XML@fenceclose\jg@lceil\let\XML@fenceclose\lceil\fi
  \ifx\XML@fenceclose\jg@rceil\let\XML@fenceclose\rceil\fi
  \ifx\XML@fenceclose\jg@lmoustache\let\XML@fenceclose\lmoustache\fi
  \ifx\XML@fenceclose\jg@rmoustache\let\XML@fenceclose\rmoustache\fi
  \ifx\XML@fenceclose\jg@lgroup\let\XML@fenceclose\lgroup\fi
  \ifx\XML@fenceclose\jg@rgroup\let\XML@fenceclose\rgroup\fi
  \ifx\XML@fenceclose\jg@uparrow\let\XML@fenceclose\uparrow\fi
  \ifx\XML@fenceclose\jg@downarrow\let\XML@fenceclose\downarrow\fi
  \ifx\XML@fenceclose\jg@Uparrow\let\XML@fenceclose\Uparrow\fi
  \ifx\XML@fenceclose\jg@Downarrow\let\XML@fenceclose\Downarrow\fi
  \ifx\XML@fenceclose\jg@updownarrow\let\XML@fenceclose\updownarrow\fi
  \ifx\XML@fenceclose\jg@Updownarrow\let\XML@fenceclose\Updownarrow\fi
}
```

Here we put each delimiter in a string command.

```
\XMLstring\jg@verbar<>&#x2016;</>
\XMLstring\jg@Verbar<>&#8214;</>
\XMLstring\jg@lfloor<>&#x230A;</>
\XMLstring\jg@rfloor<>&#x230B;</>
\XMLstring\jg@lceil<>&#x2308;</>
\XMLstring\jg@rceil<>&#x2309;</>
\XMLstring\jg@lmoustache<>&#x23B0;</>
\XMLstring\jg@rmoustache<>&#x23B1;</>
\XMLstring\jg@lgroup<>&#x3014;</>
\XMLstring\jg@rgroup<>&#x3015;</>
\XMLstring\jg@uparrow<>&#x2191;</>
\XMLstring\jg@Uparrow<>&#x21D1;</>
\XMLstring\jg@downarrow<>&#x2193;</>
\XMLstring\jg@Downarrow<>&#x21D3;</>
```

```
\XMLstring\jg@updownarrow<>&#x2195;</>
\XMLstring\jg@Updownarrow<>&#x21D5;</>
```

The Unicode standard says that U+2329 and U+232A should not be used for math. For this reason, we add U+27E8 and U+27E9 in this list.

```
\XMLstring\jg@lbra<>&#x2329;</>
\XMLstring\jg@rbra<>&#x232A;</>
\XMLstring\jg@xlbra<>&#x27E8;</>
\XMLstring\jg@xrbra<>&#x27E9;</>
```

3.2.2 Accents

The translation by Tralics of `$\vec x$` is `<mover accent='true'> <mi>x</mi> <mo>→ </mo> </mover>`. A non obvious point is how to translate this back. This idea is that we evaluate the accent in a context (defined by `\notinover`), this will set the two commands `\cur@mo@content` and `\jg@cur@acc`. Let's declare them here.

```
\let\notinover\relax
\def\cur@mo@content{TOTO}
\let\jg@cur@acc\relax
```

Processing of `<m:munderover accent=`*bool* `accentunder=`*bool*`>` *base underscript overscript* `</m:munderover>`. This is wrong. Moreover, attributes are ignored.

```
\XMLelement{m:munderover}
  { }
  {\xmlgrab}
  {\xmltexthreechildren\@firstofone\sb\sp#1}
```

Processing of `<m:mover accent=`*bool*`>` *base overscript* `</m:mover>`. We use an intermediate command. If no accent, we use `\stackrel`. Otherwise, we call a command, say `\jg@bindings`, evaluate the accent in an environment where `\notinover` is `\over`, then execute two commands `\cur@mo@content` and `\jg@cur@acc` (`\X` and `\Y` for simplicity). Evaluating the overscript should define `\X`, and evaluating `\X` should define `\Y`, in such a way that `\Y` applied to the base should put an accent on it.

```
\XMLelement{m:mover}
  {\XMLattribute{accent}{\XML@overaccent}{none}}
  {\xmlgrab}
  {\xmltextwochildren\xml@implement@over{}#1}
\def\xml@implement@over#1#2{%
 \ifx\XML@overaccent\att@true {%
   \jg@bindings
   \let\notinover\over #2\let\notinover\relax \cur@mo@content\jg@cur@acc{#1}%
 }\else\stackrel{#2}{#1}\fi}
```

Processing of `<m:munder accentunder=`*bool*`>` *base underscript* `</m:munder>`. We use an intermediate command: same idea as above. If no accent, we use `\underset`.

```
\XMLelement{m:munder}
  {\XMLattribute{accentunder}{\XML@underaccent}{none}}
  {\xmlgrab}
  {\xmltextwochildren\xml@implement@under{}#1}
\def\xml@implement@under#1#2{%
 \ifx\XML@underaccent\att@true {%
   \jg@ubindings
```

```
    \let\notinover\over #2\let\notinover\relax \cur@mo@content\jg@cur@acc{#1}%
 }\else \underset{#2}{#1}\fi}
```

Processing of <m:mo *atts*>...</m:mo>. The MathML norm says that the following attributes are recognized: form, fence, separator, lspace, rspace, stretchy, symmetric, maxsize, minsize, largeop, movablelimits, and accent. We implement only two attributes. We assume that, if form='prefix' is given, that our operator is like log or sin, and never used as an accent. Thus, we generate a \mathop with an operator font. We set \limits or \nolimits, depending on the value of the movablelimits attribute. Otherwise, some command is called.

```
\XMLelement{m:mo}
  {\XMLattribute{form}{\XML@mathmlform}{inline}%
   \XMLattribute{movablelimits}{\XML@movablelimits}{false}}
  {\xmlgrab}
  {\ifx\XML@mathmlform\att@PREFIX
     \ifx\XML@movablelimits\att@true
       \mathop{\operator@font #1}\limits
     \else
       \mathop{\operator@font #1}\nolimits
     \fi
   \else\special@mo{#1}\fi}
```

For some strange reason, the default code does not produce a less than sign. Hence we hack a bit. In the case of accent, we have two commands, let's call them \X and \Y. We define \X, so that it will define \Y. The definition is global (it must be seen by the caller of the <mo> element). In the case of grave accent, what we have here is a 7bit character, and we have to set \Y directly. In the case of 3 or 4 dots, we have to set this command.

```
\def\special@mo#1{%
     \def\jg@tck{#1}% save the argument
     \ifx\notinover\over% we cannot typeset here
        \ifx\jg@tck\jg@accgrave % strange
          \global\let\jg@cur@acc\jg@grave@acc
          \global\let\cur@mo@content\relax
        \else \ifx\jg@tck\jg@accdddot % strange
          \global\let\jg@cur@acc\dddot
          \global\let\cur@mo@content\relax
        \else \ifx\jg@tck\jg@accddddot % strange
          \global\let\jg@cur@acc\ddddot
          \global\let\cur@mo@content\relax
        \else\gdef\cur@mo@content{#1}\fi\fi\fi
     \else % typeset the argument, handle < and > in the correct way
     \ifx\jg@tck\jg@gt\string>\else
     \ifx\jg@tck\jg@lt\string<\else
     #1\fi\fi\fi}
```

Some declarations, needed above.

```
\XMLstring\jg@lt<>&lt;</>
\XMLstring\jg@gt<>&gt;</>
\XMLstring\jg@accgrave<>‘</>
\XMLstring\jg@accdddot<>&#x20DB;</>
\XMLstring\jg@accddddot<>&#x20DC;</>
```

This is the big hack. Remember that, in the case of \vec, the accent is an arrow, in fact, a Unicode character, that typesets as an arrow. What we do is to redefine \rightarrow, in order to put in \Y (in fact in \jg@cur@acc) a command that behaves like \vec. In the case of a breve accent,

the character 2D8 is defined to be `\ifmmode \u\else \textasciibreve \fi`, so that we have to redefine `\u`. In the case of the character 302, the definition is `\ifmmode \hat{}\else \^{}\fi`, so that we have to redefine `\hat` in such a way that it reads an argument, and evaluates to something that looks like `\hat`.

```
\def\jg@bindings{%
   \def\texttildelow {\global\let\jg@cur@acc\jg@tilde@acc}%
%   \def\textasciimacron {\global\let\jg@cur@acc\jg@cur@accB}%
    \def\textasciicircum{\global\let\jg@cur@acc\jg@hat@acc}
   \def\textasciicaron {\global\let\jg@cur@acc\jg@check@acc}%
   \def\u{\global\let\jg@cur@acc\jg@breve@acc}%
   \def\hat##1{\global\let\jg@cur@acc\jg@hat@acc}%
   \def\dot##1{\global\let\jg@cur@acc\jg@dot@acc}%
   \def\mathring##1{\global\let\jg@cur@acc\jg@ring@acc}%
   \def\textasciidieresis{\global\let\jg@cur@acc\jg@ddot@acc}%
   \let\JGG@orig@rarrow\rightarrow
   \let\JGG@orig@larrow\leftarrow
   \def\rightarrow{\global\let\jg@cur@acc\jg@overRarrow@acc}%
   \def\leftarrow{\global\let\jg@cur@acc\jg@overLarrow@acc}%
   \def\textoverbrace{\global\let\jg@cur@acc\overbrace}
   \def\textunderbrace{\global\let\jg@cur@acc\underbrace}
   \def\textasciiacute{\global\let\jg@cur@acc\acute}
   \def\textasciimacron{\global\let\jg@cur@acc\overline}
   \def\ring{\global\let\jg@cur@acc\mathring}
}
```

The same idea is used in the case of underaccent.

```
\def\jg@ubindings{%
\let\JGG@orig@rarrow\rightarrow
\let\JGG@orig@larrow\leftarrow
   \def\texttildelow {\global\let\jg@cur@acc\jg@tilde@acc}%
   \def\textasciimacron {\global\let\jg@cur@acc\underline}%
   \def\textasciicaron {\global\let\jg@cur@acc\jg@check@acc}%
   \def\u{\global\let\jg@cur@acc\jg@breve@acc}%
   \def\hat##1{\global\let\jg@cur@acc\jg@hat@acc}%
   \def\dot##1{\global\let\jg@cur@acc\jg@dot@acc}%
   \def\textasciidieresis{\global\let\jg@cur@acc\jg@ddot@acc}%
   \let\JGG@orig@rarrow\rightarrow
   \let\JGG@orig@larrow\leftarrow
   \def\rightarrow{\global\let\jg@cur@acc\jg@underRarrow@acc}%
   \def\leftarrow{\global\let\jg@cur@acc\jg@underLarrow@acc}%
   \def\textoverbrace{\global\let\jg@cur@acc\overbrace}
   \def\textunderbrace{\global\let\jg@cur@acc\underbrace}
   \def\jgunderline{\global\let\jg@cur@acc\underline}
}
```

This defines braces as delimiters. The plain.tex file says: N.B. { and } should NOT get delcodes; otherwise parameter grouping fails!

```
\global\delcode‘{"66308
\global\delcode‘}"67309
```

This is the original definition of the accent commands.

```
\def\jg@tilde@acc{\mathaccent"707E }
\def\jg@check@acc{\mathaccent"7014 }
```

```
\def\jg@breve@acc{\mathaccent"7015 }
\def\jg@hat@acc{\mathaccent"705E }
\def\jg@dot@acc{\mathaccent"705F }
\def\jg@ddot@acc{\mathaccent"707F }
\def\jg@grave@acc{\mathaccent"7012 }
\def\jg@ring@acc{\protect \mathaccentV {mathring}017 }
```

Note that when `\jg@overRarrow@acc` is called, `\rightarrow` does something strange. We must redefine it here, because `\rightarrowfill@` uses it.

```
\def\jg@overRarrow@acc{\let\rightarrow\JGG@orig@rarrow
  \mathpalette{\overarrow@\rightarrowfill@}}
\def\jg@overLarrow@acc{\let\leftarrow\JGG@orig@larrow
  \mathpalette{\overarrow@\leftarrowfill@}}
\def\jg@underRarrow@acc{\let\rightarrow\JGG@orig@rarrow
   \mathpalette{\underarrow@\rightarrowfill@}}
\def\jg@underLarrow@acc{\let\leftarrow\JGG@orig@larrow
  \mathpalette{\underarrow@\leftarrowfill@}}
```

3.2.3 More math

This is the main math element. Only the display attribute is taken into account. It can be inline or block. If it is block, we use brackets, otherwise parentheses. The original code defined a command `\GATHER`.

```
\XMLelement{m:math}
  {\XMLattribute{display}{\XML@mathmlmode}{inline}}
  {
     \ifx\XML@mathmlmode\att@BLOCK\[\else\(\fi
   }
  {
     \ifx\XML@mathmlmode\att@BLOCK\]\else\)\fi
  }
```

Processing of `<m:mfrac linethickness=`*A* `numalign=`*B* `denomalign=`*C* `bevelled=`*D*`>`*num denom*`</m:mfrac>`. We ignore the bevelled attribute. In the current version, we also ignore the horizontal alignment attributes.

```
\XMLelement{m:mfrac}
  {\XMLattribute{linethickness}{\XML@linethickness}{true}%
   \XMLattribute{numalign}{\XML@numalign}{center}%
   \XMLattribute{denomalign}{\XML@denomalign}{center}%
  }
  {\xmlgrab}
  {\xmltextwochildren\xml@implement@frac{}#1}
\def\xml@implement@frac#1#2{%
  \ifx\XML@linethickness\att@true\frac{#1}{#2}%
  \else \genfrac{}{}\XML@linethickness{}{#1}{#2}\fi
```

3.2.4 Tables

Processing of `<m:mtable` *atts*`>`*body*`</m:mtable>`. The MathML specification says that the following attributes are allowed: align, rowalign, columnalign, groupalign, alignmentscope, columnwidth, width, rowspacing, columnspacing, rowlines, columnlines, frame, framespacing, equalrows,

equalcolumns, displaystyle, side, and minlabelspacing. They are currently all ignored. New implementation, dated January 2005.

```
\XMLelement{m:mtable}
  {}
  {\xmlgrab }
  {\jg@start@mtable#1\jg@end@mtable }
```

Processing of <m:mtr *atts*>*body*</m:mtr>. The attributes are: rowalign, columnalign, and groupalign.

```
\XMLelement{m:mtr}
  {}
  {\xmlgrab }
  {\jg@start@mtr#1}
```

Processing of <m:mtd *atts*>*body*</m:mtd>. The attributes are: rowspan, columnspan, rowalign, columnalign, and groupalign.

```
\XMLelement{m:mtd}
 {\XMLattribute{columnalign}{\XML@mtdalign}{center}
  \XMLattribute{columnspan}{\XML@mtdspan}{1}}
 {\xmlgrab}
 {\jg@start@mtd{#1}}
```

The current implementation of math tables uses two token lists, one holds the table, the other one holds the current row. There is a command `\addto@hook` that inserts some tokens at the end of a token list; it is similar to `\gaddto@hook` without the `\global`. The command `\merge@toks` appends the current row to the table (action has to be global, because it is executed from the group that defines the row).

```
\newtoks\jg@table@toks
\newtoks\jg@row@toks
\newcommand\merge@toks[2]{%
  \expandafter\gaddto@hook\expandafter#1\expandafter{\the#2}}
\long\def\gaddto@hook#1#2{\global#1\expandafter{\the#1#2}}
```

When we see the start of a table, we initialize the token list. What we will finally evaluate is an array, declared as `*{99}{c}`. In the current implementation, it is possible to count the number of columns, and replace 99 by a better value, but this would cost more than constructing a preamble of length 99. Note that, in amsmath, matrices are declare in this way, using the counter `MaxMatrixCols`, with initial value 10, instead of the constant 99. We initialize the row token list to be empty, and `\ifStartTable` to true (this means that the next row is the first one). When the end of the array is seen, we insert the last row and the `\end{array}` in the token list, then evaluate it.

```
\newif\ifStartTable\newif\ifStartRow
\newcommand\jg@start@mtable{%
 \jg@table@toks{\begin{array}{*{99}{c}}}%
 \jg@row@toks{}%
 \StartTabletrue}
\newcommand\jg@end@mtable{%
 \merge@toks\jg@table@toks\jg@row@toks
 \addto@hook\jg@table@toks{\end{array}}%
 \the\jg@table@toks}
```

When we see the start of a row, we insert the previous row into the table; we reset it; we define the command `\ifStartRow` to true (this means that the next cell is the first in the row). In the

case where `\ifStartTable` is true, we set it to false, otherwise we add a `\\`, the row separator, in the array.

```
\newcommand\jg@start@mtr{%
  \merge@toks\jg@table@toks\jg@row@toks
  \ifStartTable
     \glbal\StartTablefalse
  \else \gaddto@hook\jg@table@toks{\\}\fi
  \jg@row@toks{}%
  \StartRowtrue}
```

The main action associated to a cell consists in putting the content in the `\jg@row@toks` token list; in the case where `\ifStartRow` is true, we set it to false, otherwise we add a `&`, the cell separator, in the token list. In `\temp`, we put the content of our cell, plus some `\hfill` in the case where alignment is left or right; if alignment is 'left', we insert an empty group after the `\hfill` (the preamble of the array for our cell is `\hfil\ignorespaces#\unskip\hfil`). In the general case, there is no need to use a temporary command: we could add the tokens directly into the token list; however, in the case where the span is greater than one, the action is a bit more complicated, in this case, we use a temporary token register.

```
\newcommand\jg@start@mtd[1]{%
   \ifStartRow
     \global\StartRowfalse
   \else\gaddto@hook\jg@row@toks{\tabcellsep}\fi
   \ifx\XML@mtdalign\att@mtd@left
      \def\temp{#1\hfill{}}%
    \else\ifx\XML@mtdalign\att@mtd@right
      \def\temp{\hfill#1}%
        \else \def\temp{#1}\fi\fi%
  \ifnum\XML@mtdspan=1\else
   \toks0=\expandafter{\temp}%
   \edef\temp{\noexpand\multicolumn{\XML@mtdspan}{c}{\the\toks0}}%
  \fi
  \expandafter\gaddto@hook\expandafter\jg@row@toks\expandafter{\temp}}
```

3.3 Other commands

3.3.1 Pictures

In this section, we define some elements that Tralics constructs when evaluating a command from the epic package. These elements are in the default namespace with 'pic-' prefix; in a future version, we will move them in another namespace.

Processing of `<picture width=`*A* `height=`*B* `xpos=`*C* `ypos=`*D*`>...</picture>`. This defines the picture environment. Translation is `\begin{picture} (A,B)(C,D)` ... `\end{picture}`.

```
\XMLelement{picture}
  {\XMLattribute{width}{\XML@width}{1}
   \XMLattribute{height}{\XML@height}{1}
   \XMLattribute{xpos}{\XML@xpos}{0}
   \XMLattribute{ypos}{\XML@ypos}{0}
  }
  {\begin{picture}(\XML@width,\XML@height)(\XML@xpos,\XML@ypos)}
  {\end{picture}}
```

Processing of `<pic-put xpos=A ypos=B>C</pic-put>`. We translate this into `\put(A,B){C}`.

```
\XMLelement{pic-put}
  {\XMLattribute{xpos}{\XML@xpos}{0}
   \XMLattribute{ypos}{\XML@ypos}{0}}
  {\xmlgrab}
  {\put(\XML@xpos,\XML@ypos){#1}}
```

Processing of `<pic-arc xpos=A ypos=B angle=C></pic-arc>`. We translate this in an obvious manner into `\arc(A,B){C}`.

```
\XMLelement{pic-arc}
  {\XMLattribute{xpos}{\XML@xpos}{0}
   \XMLattribute{angle}{\XML@angle}{0}
   \XMLattribute{ypos}{\XML@ypos}{0}}
  {\xmlgrab}
  {\arc(\XML@xpos,\XML@ypos){\XML@angle}}
```

Processing of `<pic-scaleput xpos=A ypos=B xscale=xs yscale=ys xscaley=xsy yscaley=ysy>C</pic-scaleput>`. We translate this into `\scaleput(A,B){C}`. Other attributes are put in variables named `\xscale`, `\yscale`, `\xscalex`, and `\yscalex`.

```
\XMLelement{pic-scaleput}
  {\XMLattribute{xscale}{\xscale}{1.0}
   \XMLattribute{yscale}{\yscale}{1.0}
   \XMLattribute{xscaley}{\xscaley}{0.0}
   \XMLattribute{yscalex}{\yscalex}{0.0}
   \XMLattribute{xpos}{\XML@xpos}{0}
   \XMLattribute{ypos}{\XML@ypos}{0}}
  {\xmlgrab}
  {\scaleput(\XML@xpos,\XML@ypos){#1}}
```

Processing of `<pic-thicklines/>` and `<pic-thinlines/>`. We call a command, that is executed just after the group. In the case of `<pic-linethickness size=V/>`, the command takes an argument; we have to expand the value of the attribute, put this in a global variable, and ask for that variable to be expanded after the group. In a first version, we redefined the three commands `\thinlines`, `\thicklines` and `\linethickness`. The current code is simpler.

```
\XMLelement{pic-thicklines}
  {}{}{\aftergroup\thicklines}

\XMLelement{pic-thinlines}
  {}{}{\aftergroup\thinlines}

\XMLelement{pic-linethickness}
  {\XMLattribute{size}{\XML@size}{1pt}}
  {}
  {\xdef\temp{\noexpand\linethickness{\XML@size}}\aftergroup\temp}
```

Processing of `<pic-multiput xpos=A ypos=B repeat=C dx=D dy=E>obj</pic-multiput>`. We translate this as `\multiput(A,B)(D,E){C}{obj}`.

```
\XMLelement{pic-multiput}
  {\XMLattribute{xpos}{\XML@xpos}{0}
   \XMLattribute{ypos}{\XML@ypos}{0}
   \XMLattribute{repeat}{\XML@repeat}{1}
   \XMLattribute{dx}{\XML@dx}{1}
   \XMLattribute{dy}{\XML@dy}{1}}
  {\xmlgrab}
  {\multiput(\XML@xpos,\XML@ypos)(\XML@dx,\XML@dy){\XML@repeat}{#1}}
```

Processing of <pic-bezier a1=*A1* a2=*A2* b1=*B1* b2=*B2* c1=*C1* c2=*C2* repeat=*D*/>.

```
\XMLelement{pic-bezier}
  {\XMLattribute{a1}{\XML@ai}{0}
   \XMLattribute{a2}{\XML@aii}{0}
   \XMLattribute{b1}{\XML@bi}{0}
   \XMLattribute{b2}{\XML@bii}{0}
   \XMLattribute{c1}{\XML@ci}{0}
   \XMLattribute{c2}{\XML@cii}{0}
   \XMLattribute{repeat}{\XML@repeat}{0}}
  {}
  {
   \qbezier[\XML@repeat](\XML@ai,\XML@aii)(\XML@bi,\XML@bii)(\XML@ci,\XML@cii)
  }
```

Processing of <pic-line xdir=*A* ydir=*B* width=*C*/>. The translation is `\line(A,B){C}`.

```
\XMLelement{pic-line}
  {\XMLattribute{xdir}{\XML@xdir}{0}
   \XMLattribute{ydir}{\XML@ydir}{0}
   \XMLattribute{width}{\XML@width}{0}}
  {}
  {\line(\XML@xdir,\XML@ydir){\XML@width}}
```

Processing of <pic-vector xdir=*A* ydir=*B* width=*C*/>. The translation is `\vector(A,B){C}`.

```
\XMLelement{pic-vector}
  {\XMLattribute{xdir}{\XML@xdir}{0}
   \XMLattribute{ydir}{\XML@ydir}{0}
   \XMLattribute{width}{\XML@width}{0}}
  {}
  {\vector(\XML@xdir,\XML@ydir){\XML@width}}
```

Processing of <pic-curve unit-length=*A*>*B*</pic-curve>. The translation is `\curve(B)`. There are two hacks here: one is that `\curve` modifies `\unitlength` globally, so that we have to reset it. The other hack is that `\ignorespaces` must be ignored: the argument of `\curve` is a sequence of numbers, converted to dimensions by using `\unitlength`. Between a sequence of digits and a unit of measure, spaces are allowed (but here, spaces are active, there expansion is: space plus `\ignorespaces`).

```
\newdimen\jgunitlength
\newcommand\withulength[1]{%
  {\def\ignorespaces{}%
   \jgunitlength=\unitlength \setlength\unitlength{\XML@ulength pt}%
   #1\global\unitlength\jgunitlength}}

\XMLelement{pic-curve}
  {\XMLattribute{unit-length}{\XML@ulength}{1}}
  {\xmlgrab}
  {\withulength{\curve(#1)}}
```

Processing of <pic-closecurve unit-length=*A*>*B*</pic-closecurve>.
The translation is `\closecurve(B)`. Save hack as above.

```
\XMLelement{pic-closecurve}
  {\XMLattribute{unit-length}{\XML@ulength}{1}}
  {\xmlgrab}
  {\withulength{\closecurve(#1)}}
```

Processing of `<pic-tagcurve unit-length=A> B</pic-tagcurve>`.
The translation is `\tagcurve(B)`. Save hack as above.

```
\XMLelement{pic-tagcurve}
  {\XMLattribute{unit-length}{\XML@ulength}{1}}
  {\xmlgrab}
  {\withulength{\tagcurve(#1)}}
```

Processing of `<pic-frame>X</pic-frame>`. Translation is `\frame{X}`.

```
\XMLelement{pic-frame}
 {}
 {\xmlgrab}
 {\frame{#1}}
```

Processing of `<pic-circle size=A full=B/>`. Translation is `\circle{A}` or `\circle*{A}`.

```
\XMLelement{pic-circle}
  {\XMLattribute{size}{\XML@size}{1}
   \XMLattribute{full}{\XML@full}{false}}
  {}
  {\ifx\XML@full\att@false\circle{\XML@size}\else \circle*{\XML@size}\fi}
```

Processing of `<pic-circle size=A unit-length=B/>`. Translation is `\bigcircle{A}`. We locally change `\unitlength`.

```
\XMLelement{pic-bigcircle}
  {\XMLattribute{size}{\XML@size}{1}
   \XMLattribute{unit-length}{\XML@ulength}{1}}
  {}
  {\withulength{\bigcircle{\XML@size}}}
```

The strings defined here are needed in the next command.

```
\XMLstring\att@l<>l</>
\XMLstring\att@r<>r</>
\XMLstring\att@t<>t</>
\XMLstring\att@b<>b</>
\XMLstring\att@lt<>lt</>
\XMLstring\att@lb<>lb</>
\XMLstring\att@rt<>rt</>
\XMLstring\att@rb<>rb</>
\XMLstring\att@tl<>tl</>
\XMLstring\att@bl<>bl</>
\XMLstring\att@tr<>tr</>
\XMLstring\att@br<>br</>
```

Given a string A, in `\XML@pos`, we construct a string B, that has the same characters (order is irrelevant), with category code 11. Moreover, if we have a command `\bar` and an argument `gee`, we evaluate `\bar[B]{gee}` (in fact, we construct a command that does this; this command will be evaluated after all conditionals have been closed.)

```
\def\@att@to@rtb#1#2{%
  \ifx\XML@pos\att@l \def\jg@tmp{#1[l]{#2}}
  \else\ifx\XML@pos\att@r \def\jg@tmp{#1[r]{#2}}%
  \else\ifx\XML@pos\att@t \def\jg@tmp{#1[t]{#2}}%
  \else\ifx\XML@pos\att@b \def\jg@tmp{#1[b]{#2}}%
  \else\ifx\XML@pos\att@lt \def\jg@tmp{#1[lt]{#2}}%
  \else\ifx\XML@pos\att@lb \def\jg@tmp{#1[lb]{#2}}%
  \else\ifx\XML@pos\att@rt \def\jg@tmp{#1[rt]{#2}}%
```

```
  \else\ifx\XML@pos\att@rb \def\jg@tmp{#1[rb]{#2}}%
  \else\ifx\XML@pos\att@tl \def\jg@tmp{#1[lt]{#2}}%
  \else\ifx\XML@pos\att@bl \def\jg@tmp{#1[lb]{#2}}%
  \else\ifx\XML@pos\att@tr \def\jg@tmp{#1[rt]{#2}}%
  \else\ifx\XML@pos\att@br \def\jg@tmp{#1[rb]{#2}}%
  \else \def\jg@tmp{#1{#2}}
 \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
 \jg@tmp
}
```

Processing `<pic-framebox width=`*A* `height=`*B* `position=`*C* `framed=`*D*`>`*obj*`</pic-framebox>`. The translation is `\framebox(A,B)[CC]{obj}` (it could be `\makebox` if the attribute framed is false). Here CC is the value of C, processed as indicated above.

```
\XMLelement{pic-framebox}
  {\XMLattribute{width}{\XML@width}{0}
   \XMLattribute{height}{\XML@height}{0}
   \XMLattribute{position}{\XML@pos}{}
   \XMLattribute{framed}{\XML@framed}{false}
  }
 {\xmlgrab}
 {\let\cmd\framebox\ifx\XML@framed\att@false\let\cmd\makebox\fi
  \@att@to@rtb{\cmd(\XML@width,\XML@height)}{#1}}
```

Processing `<pic-dashbox width=`*A* `height=`*B* `position=`*C* `dashdim=`*D*`> `*obj*` </pic-dashbox>`. The translation is `\dashbox{D}(A,B)[CC]{obj}`, where CC is as above.

```
\XMLelement{pic-dashbox}
  {\XMLattribute{width}{\XML@width}{0}
   \XMLattribute{height}{\XML@height}{0}
   \XMLattribute{position}{\XML@pos}{w}
   \XMLattribute{dashdim}{\XML@dashdim}{1pt}
  }
 {\xmlgrab}
 {\@att@to@rtb{\dashbox{\XML@dashdim}(\XML@width,\XML@height)}{#1}}
```

Processing of `<pic-oval xpos=`*A* `ypos=`*B* `specs=`*C*`>`*obj*`</pic-oval>`.
The translation is `\oval(A,B)[CC]{obj}`, where CC is as above.

```
\XMLelement{pic-oval}
  {\XMLattribute{xpos}{\XML@xpos}{0}
   \XMLattribute{specs}{\XML@pos}{}
   \XMLattribute{ypos}{\XML@ypos}{0}}
  {\xmlgrab}
  {\@att@to@rtb{\oval(\XML@xpos,\XML@ypos)}{#1}}
```

3.3.2 Titlepage

We found no easy way to describe the title page of the Raweb using XSL/Format; thus additional elements were invented and described here. The `\ra@atxy` command adds a box (that occupies no space) at a give position in the page (the position is absolute).

```
\newbox\ra@atxybox
\def\ra@atxy(#1,#2)#3{\global\setbox\ra@atxybox=\hbox
 {\unhbox\ra@atxybox
  \vtop to 0pt{\kern #2\hbox to 0pt{\kern #1\relax #3\hss}\vss}}}
```

We use `\@texttop` (a command that does nothing)[1] for insertion of our box. After that, we kill our command. We have to shift horizontally by 1in plus `\hoffset` and the current margin, vertically by 1in, plus `\voffset` plus `\headheight` plus `\headsep` plus `\topmargin`.

```
\def\ra@useatxy{{%
  \let\@themargin\oddsidemargin
  \vtop to 0pt{\kern-\headsep \kern-\topmargin \kern-\headheight
               \kern-1in \kern-\voffset
    \hbox to 0pt{\kern-\@themargin \kern-1in \kern-\hoffset
    \unhbox\ra@atxybox \hss}\vss}}%
    \global\let\@texttop\relax}
```

The command `\firstchar` is nowadays useless. In the times when Inria themes were 1A, 1B, 2A, 2B etc, we used it to remove the letter. Nowadays themes are NUM, COG, etc, and we hope people give only the theme.[2] We insert the logo, and, above it, something that looks like
———— THEME NUM ————.

```
\def\foratheme#1{\vskip8cm \vfil
  \ra@atxy(74mm,175mm) {\hbox to 70mm{%
       \hrulefill\hspace{8mm}
       \def\firstchar##1##2\relax{##1##2} % ducon
       \href{http://www.inria.fr/recherche/equipes/listes/theme%
         _\firstchar#1\relax.en.html}{THEME \uppercase{#1}}%
       \hspace{8mm}\hrulefill}}}
```

Start of the title page.

```
\def\foinria{%
  \ra@atxy(7.8cm,2.5cm){\includegraphics[width=5.7cm]{Logo-INRIA-couleur}}%
  \ra@atxy(55mm,173mm){\includegraphics{LogoRA\ra@year}}%
 \setbox0\hbox to 14cm{%
     \noindent\hskip3cm\hfill
     {\fontencoding{T1}\fontfamily{ptm}\fontseries{m}%
   \fontshape{n}\fontsize{10pt}{12pt}\selectfont
\href{http://www.inria.fr}{INSTITUT NATIONAL DE RECHERCHE EN %
     INFORMATIQUE ET EN AUTOMATIQUE}}%
      \hskip-5cm\hfill}%
  \null\vskip0.7cm \leavevmode\hskip-3.5cm\box0\null\vskip2cm\vfil}
```

This is a hack: we just want `\cleardoublepage`. This is not used anymore.

```
\XMLelement{cleardoublepage}
 {} {\cleardoublepage} {}
```

This is a hack: we want a rule below the page headings. We found no other way. This is not used anymore.

```
\XMLelement{pagestylehrule}
 {} {\hbox to0pt{\rule[-1ex]{\textwidth}{.03cm}\hss}} {}
```

3.3.3 Images

For the HTML version of the Raweb, we convert all math formulas into images. The idea is to construct a XML file that contains a `<formula>` for each object to convert; the id attribute is the number of the image. Here we translate `<formula id=A>math</formula>`.

[1] This is redefined by `\raggedbottom` to be empty; you have to find a different trick if you want to insert something at a given position using this code as a model.

[2] and not the subtheme; we could use a command that picks up the first three characters.

```
\XMLelement{formula}
 {\XMLattribute{id}{\XML@formid}{none}}
 {\@inlinemathA{\XML@formid}}
 {\@inlinemathZ}
```

For some strange reasons, the fotex.sty file says that eps files should be included by converting them into pdf; this works for pdfTEX, but not for LATEX. We make sure here that this rule is never applied. We redefine `\normalsize`, in the same way as `latex2html`.

```
\newbox\sizebox
\AtBeginDocument{
\@namedef{Gin@rule@.eps}#1{{eps}{.eps}{#1}}
\let\realnormalsize=\relax
\def\adjustnormalsize{%
  \def\normalsize{\mathsurround=0pt \realnormalsize
   \parindent=0pt\abovedisplayskip=0pt\belowdisplayskip=0pt}%
  \def\phantompar{\csname par\endcsname}%
  \normalsize}}
```

The action at the start of a formula is the following: we remember the id somewhere, we start a new page, and we construct a box; this box will start with a little space, it has a strut.

```
\def\@inlinemathA#1{%
  \gdef\[{$\displaystyle}\gdef\]{$} % hack...
  \xdef\@mathenv{#1}%
  \adjustnormalsize \newpage\clearpage
  \setbox\sizebox=\hbox\bgroup\kern.05em
  \vrule height1.5ex width0pt }
```

Assume that x is (a bit more than) the maximum of the height and depth of `\sizebox`. We modify the height and depth to be x; we insert a vertical rule of height x and depth x. After this piece of code has been executed, the image should be vertically centered (otherwise, the bottom should be aligned on the baseline).

```
\def\@centerinlinemath{%
  \dimen1=\ifdim\ht\sizebox<\dp\sizebox \dp\sizebox\else\ht\sizebox\fi
  \advance\dimen1by.5pt \vrule width0pt height\dimen1 depth\dimen1
 \dp\sizebox=\dimen1\ht\sizebox=\dimen1\relax}
```

This is done at the end. The purpose of these `\expandafter` is unclear (this code comes from `latex2html`). The action is the following: we terminate the box, add some code that can possibly center the image, and print the dimensions on the log file. After that, we add an horizontal and a vertical rule, so that dvips can compute the bounding box. We add some glue, and we start a new page.

```
\def\@inlinemathZ{%
  \egroup\expandafter\ifdim\dp\sizebox>0pt %
  \expandafter\@centerinlinemath\fi
  \typeout{l2hSize %
   :\@mathenv:\the\ht\sizebox::\the\dp\sizebox::\the\wd\sizebox.}
  \hbox{\vrule width.1em\kern-.05em\vtop{\vbox{%
  \kern.1em\kern0.6 pt\hbox{\hglue.17em\copy\sizebox\hglue0.6 pt}}\kern.3pt%
  \ifdim\dp\sizebox>0pt\kern.1em\fi \kern0.6 pt%
  \ifdim\hsize>\wd\sizebox \hrule depth1pt\fi}}%
  \ifdim\ht\sizebox<\vsize
      \ifdim\wd\sizebox<\hsize\expandafter\hfill\fi \expandafter\vfill
      \else\expandafter\vss\fi
```

```
  \clearpage
}
```

3.3.4 Misc

We want TeX and LaTeX to typeset properly.

```
\XMLelement{TeX}
  {}{\TeX{}}{}

\XMLelement{LaTeX}
  {}{\LaTeX{}}{}
```

We use this for producing little images.

```
\XMLelement{preview}
  {}
  {\begin{preview}}
  {\end{preview}}
```

Some attribute definitions.

```
\XMLstring\att@true<>true</>
\XMLstring\att@false<>false</>
\XMLstring\jg@OverBrace<>&#xFE37;</>
\XMLstring\jg@UnderBrace<>&#xFE38;</>
\XMLstring\jg@OverBar<>&#xAF;</>
\XMLstring\jg@UnderBAr<>&#x332;</>
\XMLstring\att@mtd@left<>left</>
\XMLstring\att@mtd@right<>right</>
\XMLstring\att@mtd@center<>center</>
\XMLstring\att@none<>none</>
\XMLstring\att@dzero<>0</>
\XMLstring\att@done<>1</>
\XMLstring\att@dtwo<>2</>
\XMLstring\att@mathml@rm<>rm</>
\XMLstring\att@BLOCK<>block</>
\XMLstring\att@PREFIX<>prefix</>
\XMLstring\att@EQUATION<>equation</>
```

Some commands must be redefined at start of document. Note: the Unicode character 3F5 is 'greek unate epsilon symbol', we translate it as ϵ, the character 3B5 'greek small letter epsilon' is translated as ε.

```
\AtBeginDocument{
  \UnicodeCharacter{x332}{\jgunderline}
  \UnicodeCharacter{x3F5}{\epsilon}
  \UnicodeCharacter{x3B5}{\varepsilon}
  \let\downslopeellipsis\ddots
  \mathchardef\Rightarrow="3229
  \let\@texttop\ra@useatxy
  \let\XURL\relax
  \selectlanguage{english}
  \let\@item\jg@item
}
```

3.4 The fotex.cfg file

The fotex.cfg file contains the definitions of some elements. Two of them are used by the Raweb. The action associated to these elements is shown above.

```
633 \XMLelement{fo:RATHEME}
634   {}
635   {\xmlgrab}
636   {\foratheme{#1}}
637
638 \XMLelement{fo:INRIA}
639   {\XMLattribute{year}{\ra@@year}{2001}}
640   {}
641   {\xdef\ra@year{\ra@@year}\foinria}
```

Chapter 4

Interpreting XSL/Format in TeX

We describe here the content of two files: fotex.xmt and fotex.sty. The fotex.xmt file contains definitions of elements, while the fotex.sty file contains some commands. Both files start like this

```
% Copyright 2002 Sebastian Rahtz/Oxford University
%      <sebastian.rahtz@oucs.ox.ac.uk>
%
% Permission is hereby granted, free of charge, to any person obtaining
% a copy of this software and any associated documentation files (the
% ``Software''), to deal in the Software without restriction, including
% without limitation the rights to use, copy, modify, merge, publish,
% distribute, sublicense, and/or sell copies of the Software, and to
% permit persons to whom the Software is furnished to do so, subject to
% the following conditions:
% The above copyright notice and this permission notice shall be included
% in all copies or substantial portions of the Software.

% Includes fixes from Tomas Bures <ghort@pauline.vellum.cz>
%                     Yura Zotov <yznews@hotbox.ru>
%                     Anton V. Boyarshinov <boyarsh@ru.echo.fr>
```

The fotex.xmt file has a comment that says it is version #90, while the style file has this:

```
\ProvidesPackage{fotex}[2002/06/25: version 1.17.
   support for XSL formatting, S Rahtz]
```

We added this for the Raweb. The fotex-add.sty file loads the following packages: graphics, multicol, rotating, curves, array, amsmath, longtable, url, ulem, color, times, mlnames, unicode, marvosym, ipa, ifsym, ucharacters, nameref, hyperref, and raweb-uni. This last package defines (or redefines) some Unicode characters. Some packages, like ifsym, were not in the original style file, they were added because providing access to some fonts.

```
\RequirePackage{fotex-add}
```

The fotex.xmt file starts with these two lines, it is followed by declarations of attributes, strings and elements. The namespace number of `fo` is 5 (see below), that of `fotex` is 6.

```
\DeclareNamespace{fotex}{http://www.tug.org/fotex}
\DeclareNamespace{fo}{http://www.w3.org/1999/XSL/Format}
```

Handling the `<fo:root>` element is trivial, but some magic is implied. Let's describe it. The main TeX file should define the `\xmlfile` command to be the name of the XML file to process, and input the xmltex.tex file. The last action of this file is to input xmltex.cfg, `\jobname`.cfg and the XML file. The file xmltex.cfg typically defines a catalog similar to the one shown in section 2.7.

It defines the XSL/Format namespace (it assigns the number 5 to it) and says that the definitions are to be found in the fotex.xmt file. It may define other things. The `\jobname.cfg` file can also define some commands.

The `\xmlfile` file is read using XML syntax. It typically starts with a `<?xml?>` declaration (that defines the encoding, say UTF-8). In the case of the Raweb, it is followed by a `<fo:root>` element. This one contains a `<fo:layout-master-set>`, followed by some `<fo:static-content>` elements and some `<fo:page-sequence>` elements. The root element has three attributes, of the form xmlns:XX=YY. For each attribute, the code line 388 (page 21) is executed. One of the attributes has the name xmlns:fo and its value is the XSL/Format URI. This URI is the same as the declaration above (on line 20 in this section), so that the 'fo' in `<fo:root>` is the same namespace in the XML file and the xmt file. After all attributes have been read, the code on line 413 is executed. In particular, the 'fo' in `<fo:root>` is evaluated now (line 416, command `\XML@ns`, the value is 5). The command `\XML@checkknown` has as side effect to call `\inputonce` with, as argument, the file fotex.xmt, as explained in the catalogue. This has as effect to load the fotex.xmt file. After that, we know how to handle the root element. The first action is to evaluate the attributes. Almost all attributes are global. The norm[1] says: there is one attribute media-usage, in what follows, we shall assume that its value is '`paginate`'. The action is as follows: we define the documentclass to something non-trivial but not too complicated, and load the fotex package.

```
\XMLelement{fo:root}
  {}
  {\documentclass{article}
   \usepackage{fotex}
```

We are now ready to evaluate everything. We start with `\begin{document}` plus some action (empty pagestyle, default language). When we see `</fo:root>` we evaluate `\end{document}`: this will stop everything.

```
   \begin{document}
   \pagestyle{empty}
   \FOSetHyphenation
   }
  {\end{document}}
```

Bootstrap code: We use here `\XMLNSA` as a short name for `\XMLnamespaceattribute` (and the same for `\XMLNSAX`). Remember that this takes four arguments. In the line that follows, the effect is the following: for every element in the '`fo`' namespace, the value of the language trait is stored in `\FOlanguage`; the default value is '`\inherit`', this means that it is the same as the value of the father. The initial value is '`none`'. In the case of the hyphenate trait, the initial value is '`false`', the default value is '`\inherit`'. Note: the mechanism works only if attributes are declared before elements; the fotex.xmt file starts with the list of all attributes, in alphabetic order; we prefer give the definition after first use.

```
\XMLNSAX{fo}{language}{\FOlanguage}{\inherit}
\XMLNSAX{fo}{hyphenate}{\FOhyphenate}{\inherit}
\gdef\FOlanguage{none}
\gdef\FOhyphenate{false}
\def\LastLanguage{(undefined)}
\selectlanguage{english}
```

In the case where hyphenation is desired, this piece of code is assumed to switch to the right language, it sets `\hyphenpenalty` to some value (typically 50), otherwise to infinity.

```
\def\FOSetHyphenation{%
  \ifx\FOhyphenate\att@true
```

[1] URL is http://www.w3.org/TR/xsl/slice6.html, see also slices 1 to 7.

```
      \LoadLanguage{\FOlanguage}%
      \hyphenpenalty=\exhyphenpenalty
  \else
      \hyphenpenalty=10000
  \fi}
```

We put in `\newL` a canonical version of the language, say 'FR', then evaluate `\L@FR`, unless the language is the same. We remember in `\LastLanguage` the language. The file mlnames.sty defines a lot of languages[2].

```
\def\LoadLanguage#1{%
  \begingroup\utfeight@protect@chars\xdef\newL{#1}\endgroup
  \ifx\newL\LastLanguage
  \else
    \csname L@\newL\endcsname
  \fi
  \edef\LastLanguage{\newL}}
```

4.1 Mathematics

The elements defined in this paragraph are not defined by XSL/Format. In fact, they are not used by the Raweb.

Typesetting of `<fotex:inlinemath>`*math*`</fotex:inlinemath>`. We evaluate `\(`*math*`\)`.

```
\XMLelement{fotex:inlinemath}
   {} {\(}  {\)}
```

Typesetting of `<fotex:equation>`*math*`</fotex:equation>`. The evaluation of the *math* is in an `equation` environment.

```
\XMLelement{fotex:equation}
   {}
   {\begin{equation}}
   {\end{equation}}
```

Typesetting of `<fotex:displaymath>`*math*`</fotex:displaymath>`. The evaluation of the *math* is a `displaymath` environment.

```
\XMLelement{fotex:displaymath}
   {}
   {\begin{displaymath}}
   {\end{displaymath}}
```

Typesetting of `<fotex:eqnarray>`*math*`</fotex:eqnarray>`. The evaluation of the *math* is a `gather*` environment. This is an amsmath environment, that requires an explicit `\end` tag, so that we need to grab it.

```
\XMLelement{fotex:eqnarray}
 {}
 {\xmlgrab}
 {\begin{gather*}#1\end{gather*}}
```

Handling of `<fotex:subeqn>`*math*`</fotex:subeqn>`. There is some action, that consists of evaluating the math, and a double backslash; this end-of-row marker must be executed outside the group defined by `\xmlgrab`, thus the `\aftergroup`. If there is a label (i.e. an id attribute in `\FOid`), we execute the `\label` command[3], otherwise, the equation will have no number.

[2] The English language is selected by: GB, US, gb, us, en, and none. But not EN.

[3] I don't understand this `\temp` stuff.

```
\XMLelement{fotex:subeqn}
  {}
  {\xmlgrab}
  {\ifx\FOid\@empty
     \gdef\w@t{#1\nonumber\\}
   \else
     \gdef\w@t{#1\label{\temp}\\}
   \fi
   \aftergroup\w@t}
```

4.2 Page masters

The children of `<fo:root>` are: a `<fo:layout-master-set>`, an optional `<fo:declarations>`, and some `<fo:page-sequence>`. The `<fo:declarations>` has a sequence of `<fo:color-profile>` elements as children, that define color profiles. These are currently unimplemented.

```
\XMLelement{fo:color-profile}{}{}{}
\XMLelement{fo:declarations}{}{}{}
```

The children of `<fo:layout-master-set>` are either `<fo:simple-page-master>` (that define the layout of a page) or `<fo:page-sequence-master>` (that define which simple page masters are to be used). We define in this section how simple-page-masters are handled. Each such object has a name, which is in the master-name trait. For the Raweb, we define eight such masters, named: simple1, left1, right1, first1, simple2, left2, right2, first2. Here the digit at the end indicates the number of columns of text on the page. The names left and right stand for even or odd pages (odd pages are on the right, even pages on the left). We have a special case for simple ('blank' pages) and first pages. Each master has a reference-orientation. This will be ignored; it has also a writing-mode, that will be ignored. We assume that it is 'lr-tb', meaning: inline components and text within a line are written left-to-right; lines and blocks are placed top-to-bottom. With this convention, we have before=top, after=bottom, start=left and end=right. The page is divided into five regions, four of them are called region-before, region-after, region-start and region-end (they correspond to the header, the footer, left margin, right margin). The content of these is 'static', said otherwise, it is the same for each page, except that it can contain the current page number. These regions have an extent (vertically, or horizontally). Whether the upper-left corner is part of the region-before or region-start depend on the precedence of the region-before. The four regions mentioned above surround the region-body. Both the page and the region-body have margins. How these margins and extents define the size of the body is unclear to me. For instance, the Raweb defines one page master with the following attributes: master-name='left1', page-width='210mm', page-height='297mm', margin-top='75pt', margin-bottom='100pt', margin-left='80pt', margin-right='80pt'. With these values the text width is near 15cm, and the text height is 21cm (in fact, after the text, we have some space, an empty footer, and the margin, i.e. 24pt, 12pt and 100pt, a total of nearly 5cm).

We declare here some attributes (margins are declared later).

```
\XMLNSA{fo}{page-width}{\FOpagewidth}{auto}
\XMLNSA{fo}{page-height}{\FOpageheight}{auto}
\XMLNSAX{fo}{master-name}{\FOmastername}{}
\XMLNSAX{fo}{extent}{\FOextent}{0.0pt}
```

Action is trivial: we just evaluate the content.

```
\XMLelement{fo:layout-master-set}
  {} {} {}
```

This will be used twice. The idea is that typesetting a page uses "traits", that are computed from attributes; all four margins define a single trait, that can be defined by a single attribute ('margin')

or a sequence of four attributes, or can be inherited. We shall see later that the equivalent of TEX glue is a complicated trait.

```
\def\jg@expandmargins{%
    \ifx\FOmargin\@empty\relax\else
      \let\FOmarginright\FOmargin
      \let\FOmarginleft\FOmargin
      \let\FOmargintop\FOmargin
      \let\FOmarginbottom\FOmargin
     \fi}
```

The content of a `<fo:simple-page-master>` is a definition of each of the five regions mentioned above. In this version, we ignore the start and end regions: we assume that they are blank and have zero extent. Only region-body is mandatory, so that we provide a default for region-before and region-after. After this element has been completely evaluated, six commands are defined: assume that the master name is 'left1'; we define `\left1:before`, `\left1:after`, to be names of regions, `\left1:before-extent`, `\left1:after-extent`, their extents; there is also `\left1:B` (instead of 'B', the name of the body region should be used, it is 'xsl-region-body', see below), and `\Atomic:left1`. This last command is defined here: it contains the size of the page and the max values of the margins of the page and body. As mentioned above, the sum of the vertical margins are used, so that it is unclear why one variable is replaced by the maximum. In our cases, outer margins are larger than inner margins, so that the tests are false, and no variable is changed.

```
\XMLelement{fo:simple-page-master}
  {}
  {\let\FOMaster\FOmastername
    \jg@expandmargins
    \ifx\FOpagewidth\att@auto\edef\FOpagewidth{\paperwidth}\fi
    \ifx\FOpageheight\att@auto\edef\FOpageheight{\paperheight}\fi
    \expandafter\xdef\csname\FOMaster:after\endcsname{DummyRegion}
    \expandafter\xdef\csname\FOMaster:before\endcsname{DummyRegion}
    \expandafter\xdef\csname\FOMaster:before-extent\endcsname{\FOextent}
    \expandafter\xdef\csname\FOMaster:after-extent\endcsname{\FOextent}
    }
  {
 \ifdim\InnerTopMargin>\FOmargintop\def\FOmargintop{\InnerTopMargin}\fi
 \ifdim\InnerBottomMargin>\FOmarginbottom\def\FOmarginbottom{\InnerBottomMargin}\fi
 \ifdim\InnerRightMargin>\FOmarginright\def\FOmarginright{\InnerRightMargin}\fi
 \ifdim\InnerLeftMargin>\FOmarginleft\def\FOmarginleft{\InnerLeftMargin}\fi
    \begingroup
     \utfeight@protect@chars
    \expandafter\xdef\csname Atomic:\FOMaster\endcsname{
     \MasterTopMargin\FOmargintop
     \MasterBottomMargin\FOmarginbottom
     \MasterRightMargin\FOmarginright
     \MasterLeftMargin\FOmarginleft
     \paperwidth\FOpagewidth
     \paperheight\FOpageheight
    }
   \endgroup
   }
```

These lines are from fotex.sty. The quantities `\hoffset` and `\voffset` are the opposite of the TEX offsets. The default paper width is strange.

```
\paperwidth211mm
\paperheight297mm
\hoffset-1in
\voffset-1in
```

The <fo:region-before> element is empty, it has some traits. We only consider region-name and extent. In the example given above, attributes are region-name='xsl-region-before-left' and extent='12pt'. Each region has a default name, it is not used here. The code defines two global commands. For instance, this could define \left1:before to the name shown above (in the case where the master name is 'simple1' or 'simple2' the default name is used).

```
\XMLelement{fo:region-before}
  {}
  {\ifx\FOregionname\@empty \def\FOregionname{xsl-region-before}\fi
   \begingroup
     \utfeight@protect@chars
     \expandafter\xdef\csname\FOMaster:before\endcsname{\FOregionname}
     \expandafter\xdef\csname\FOMaster:before-extent\endcsname{\FOextent}
   \endgroup
  }
  {}
```

Page footer. This is handled as above.

```
\XMLelement{fo:region-after}
  {}
  {\ifx\FOregionname\@empty \def\FOregionname{xsl-region-after}\fi
   \begingroup
     \utfeight@protect@chars
     \expandafter\xdef\csname\FOMaster:after\endcsname{\FOregionname}
     \expandafter\xdef\csname\FOMaster:after-extent\endcsname{\FOextent}
   \endgroup
  }
  {}
```

These are not implemented.

```
\XMLelement{fo:region-start}{}{}
\XMLelement{fo:region-end}{}{}
```

The <fo:region-body> is empty; it has traits as the region-before. In the example of 'left1' they are the following: margin-bottom='24pt', margin-top='24pt' (the default value for the two other margins is zero; no extent can be given here). The default value of the region-name is used, because all pages have the same layout.

There are possibly other attributes, but most of them are not implemented. As explained above, we store somewhere the margins so that the page-master can use them. We also store in \left1:B (where 'left1' is the name of the master, and B the name of the region), the four following quantities: top margin, bottom margin, column-count and column-gap: this is because the page can contain more than one column of text, and we have to specify this number and the distance between the columns.

```
\XMLelement{fo:region-body}
  {}
  {
  \jg@expandmargins
  \xdef\InnerBottomMargin{\FOmarginbottom}
  \xdef\InnerTopMargin{\FOmargintop}
  \xdef\InnerLeftMargin{\FOmarginleft}
```

```
  \xdef\InnerRightMargin{\FOmarginright}
  \ifx\FOregionname\@empty \def\FOregionname{xsl-region-body} \fi
  \begingroup
     \utfeight@protect@chars
      \expandafter\xdef\csname\FOMaster:\FOregionname\endcsname
        {\FOcolumngap|\FOcolumncount|\FOmarginbottom|\FOmargintop|}
  \endgroup
  }
 {}
```

This will be used later. It grabs the four values saved by the procedure above.

```
\def\Pass#1\\{\expandafter\@Pass#1}
\def\@Pass#1|#2|#3|#4|{%
 \columnsep=#1
 \def\NColumns{#2}%
 \def\Marginbottom{#3}%
 \def\Margintop{#4}%
 }
```

Some declarations.

```
\XMLNSAX{fo}{region-name}{\FOregionname}{}
\XMLNSAX{fo}{master-reference}{\FOmasterreference}{}
\XMLNSAX{fo}{column-gap}{\FOcolumngap}{12.0pt}
\XMLNSAX{fo}{column-count}{\FOcolumncount}{1}
\def\NColumns{1}
```

4.3 Page sequences

A `<fo:page-sequence-master>` is an element that has a single trait master-name; its content defines the page sequence with that name, according to its single child, that can be one of three possibilities. The Raweb defines the following: twoside1nofirst, twoside1, oneside1, twoside2, oneside2, but uses only the second one. The meaning is the following: all even pages are the same, as well as all odd pages; the first page is special. If you look very closely, there are three pages numbered 2. At least one of them is empty; but this is a kludge. In fact, the initial page (the title page) has no headings, it is of type First; is it followed by an empty page, generated by `<cleardoublepage/>`. This page is followed by a second page sequence that contains the table of contents; this page sequence is followed by the text; this is a page sequence that starts on a right page, numbered one. The first two page sequences should defined force-page-count to be end-on-even, rather than using this kludge. The behavior will change in 2005.

Action: we just remember the name, for use by children.

```
\XMLelement{fo:page-sequence-master}
  {}
  {
   \let\Granpa\FOmastername
  }
  {}
```

We define here `<fo:single-page-master-reference>`; this defines a single page. We have an implementation problem: the fo specifications are too complex to match those of TEX, so that we use only 4 types of pages, named Even, Odd, First, Blank. The original fotex.sty did define but not use Blank. We changed this. In what follows, we shall use 'twoside1' as the name of our page-sequence-master (the value of `\Granpa`), although the name is not adequate for a single

page sequence. In any case, we define `\First:twoside1`, `\Blank:twoside1`, `\Odd:twoside1`, `\Even:twoside1`, to be the same: the value of the master-reference-trait. This is the name of a page-master, for instance 'left1'.

```
\XMLelement{fo:single-page-master-reference}
  {}
  {
   \begingroup
   \utfeight@protect@chars
    \expandafter\xdef\csname First:\Granpa\endcsname{\FOmasterreference}
    \expandafter\xdef\csname Blank:\Granpa\endcsname{\FOmasterreference}
    \expandafter\xdef\csname Odd:\Granpa\endcsname{\FOmasterreference}
    \expandafter\xdef\csname Even:\Granpa\endcsname{\FOmasterreference}
   \endgroup
   }
  {}
```

We define here `<fo:repeatable-page-master-reference>`; this defines a sequence (of some length) of pages. The length is ignored. Thus the action is as above.

```
\XMLelement{fo:repeatable-page-master-reference}
  {}
  {
   \begingroup
    \utfeight@protect@chars
   \expandafter\xdef\csname Blank:\Granpa\endcsname{\FOmasterreference}
   \expandafter\xdef\csname First:\Granpa\endcsname{\FOmasterreference}
   \expandafter\xdef\csname Odd:\Granpa\endcsname{\FOmasterreference}
   \expandafter\xdef\csname Even:\Granpa\endcsname{\FOmasterreference}
   \endgroup
  }
  {}
```

The content of `<fo:repeatable-page-master-alternatives>` is a sequence of conditional page master references. The recommendations specify how to chose them, but we use another algorithm.

```
\XMLelement{fo:repeatable-page-master-alternatives}
  {} {} {}
```

The `<fo:conditional-page-master-reference>` may define one of `\First:twoside1`, etc., to the value of the master-reference-trait trait. This is the name of a page-master, for instance 'left1'. The conditions are defined by the following traits: page-position which can be first, last[4], rest, or any.

Here 'rest' means any page that is neither the first nor the last in a sequence; blank-or-not-blank can be blank, not-blank, or any (a blank page is a forced page that contains no text); odd-or-even can be odd, even or any. The parity of the page number is considered (If there are two consecutive pages numbered one, they are both odd; thus, this is not the same as left-or-right). What we do here is to define one and only one command. For instance, we could define `\Even:twoside1` to be '`right1`'.

```
\XMLelement{fo:conditional-page-master-reference}
  {}
```

[4] Is it possible to know, in TEX, if a given page is the last in a sequence? Maybe, we could modify the `\output` routine, so that the `\clearpage` on line 281 will output some pages, the last one being special. The situation is complicated by the fact that lines 285 ans 288 may output empty pages in the sequence. The `\clearpage` on line 297, together with the `\newpage` on lines 301 and 304 are also candidates for producing the last page in the sequence.

```
204    {
205      \begingroup
206        \utfeight@protect@chars
207    \ifx\FOoddoreven\att@even
208        \expandafter\xdef\csname Even:\Granpa\endcsname{\FOmasterreference}
209    \else
210      \ifx\FOoddoreven\att@odd
211        \expandafter\xdef\csname Odd:\Granpa\endcsname{\FOmasterreference}
212      \else
213        \ifx\FOpageposition\att@first
214         \expandafter\xdef\csname First:\Granpa\endcsname{\FOmasterreference}
215        \else
216          \ifx\FOblankornotblank\att@blank
217          \expandafter\xdef\csname Blank:\Granpa\endcsname{\FOmasterreference}
218          \else
219           \expandafter\xdef\csname Odd:\Granpa\endcsname{\FOmasterreference}
220          \fi
221        \fi
222      \fi
223    \fi
224    \endgroup
225    }
226   {}
```

Declarations of attributes used here.

```
227 \XMLNSAX{fo}{page-position}{\FOpageposition}{any}
228 \XMLNSAX{fo}{odd-or-even}{\FOoddoreven}{any}
229 \XMLNSAX{fo}{blank-or-not-blank}{\FOblankornotblank}{any}
230 \XMLstringX\att@even<>even</>
231 \XMLstringX\att@odd<>odd</>
```

A `<fo:page-sequence>` is linked via its master-reference trait to a page master or a page sequence master. Its children are some `<fo:static-content>` elements (they define the content of the regions defined by the page master, except for the region-body) and a `<fo:flow>` that defines the content of the pages. Some important attributes are: initial-page-number that indicates the number of the first page (it can be auto-even or auto-odd, this means that the first page number should be even or odd; it may imply the use of a blank page). The value of force-page-count can be end-on-odd or end-on-even (this may insert a final blank page). The language can be defined for a page sequence, a block or a character. The action is to evaluate the language trait via `\FOSetHyphenation` and to put the others in variables. Some traits are unused: country, letter-value, grouping-separator, grouping-size, format[5]. In the case of the Raweb, the attributes of the main page sequence are format=‘1’, text-align=‘justify’, hyphenate=‘true’, language=‘en’, initial-page-number=‘1’, master-name=‘twoside1’.

```
232 \XMLelement{fo:page-sequence}
233   {}
234   {\let\CurrentPageMaster\FOmasterreference
235    \let\pendingID\FOid
236    \let\PageNumber\FOinitialpagenumber
237    \let\ForcePage\FOforcepagecount
238    \FOSetHyphenation
239    \LoadLanguage{\FOlanguage}
```

[5]This last one explains how page numbers are typeset; see later.

```
240    }
241   {}
```

Attributes.

```
242 \XMLNSAX{fo}{initial-page-number}{\FOinitialpagenumber}{auto}
243 \XMLNSAX{fo}{force-page-count}{\FOforcepagecount}{auto}
244 \XMLstringX\att@autoodd<>auto-odd</>
245 \XMLstringX\att@autoeven<>auto-even</>
246 \XMLstringX\att@endonodd<>end-on-odd</>
247 \XMLstringX\att@endoneven<>end-on-even</>
```

You can say `<fo:static-content>`, with an attribute flow-name whose value is something like xsl-region-before-right. This is the region-name of some page master (used by the current page sequence). Logically, given the name, it should define the header of odd pages. The content is a sequence of blocks, that will be typeset later (on one or more pages). The action is to call a command defined below. A priori, the result should be used only by the flow that is a sibling of this element, but this is hard to do in TEX.

```
248 \XMLelement{fo:static-content}
249  {}
250  {\xmlgrab}
251  {\protectCS\FOflowname
252   \FOSetStatic{#1}{\FOflowname}}
```

The action associated to `<fo:static-content>` is to grab some parameters.

```
253 \def\FOSetStatic{\expandafter\@SetStatic\FOtextindent\\}
254 \def\@SetStatic#1\\{\expandafter\@@SetStatic\FOfontsize\\{#1}}
255 \def\@@SetStatic#1\\#2{\expandafter\@@@SetStatic\FOfontweight\\{#1}{#2}}
256 \def\@@@SetStatic#1\\#2#3{\expandafter
257     \@@@@SetStatic\FOfontvariant\\{#1}{#2}{#3}}
258 \def\@@@@SetStatic#1\\#2#3#4{\expandafter
259    \@@@@@SetStatic\FOfontstyle\\{#1}{#2}{#3}{#4}}
260 \def\@@@@@SetStatic#1\\#2#3#4#5{\expandafter
261    \@@@@@@SetStatic\FOfontstretch\\{#1}{#2}{#3}{#4}{#5}}
262 \def\@@@@@@SetStatic#1\\#2#3#4#5#6{\expandafter
263     \@@@@@@@SetStatic\FOfontfamily\\{#1}{#2}{#3}{#4}{#5}{#6}}
```

This code globally defines `\Static:xsl-region-before-right` (or a command like this) whose effect is to execute the content of the `<fo:static-content>` in a TEX group, where some parameters are defined. These parameters are: font family, font stretch, font style, font variant, font weight, font size, textindent (unused, but explicitly set to zero in the main block of the static content), (arguments `#1` to `#7`). Argument `#8` is the content, argument `#9` is the name. The switch `\ifFOinOutput` is set to true while evaluating the content. The `\XML@parent` command is locally set to empty. See explanations later.

```
264 \def\@@@@@@@SetStatic#1\\#2#3#4#5#6#7#8#9{%
265    \expandafter\gdef\csname Static:#9\endcsname{%
266      {%
267       {\def\XML@parent{}\global\FOinOutputtrue
268        \def\FOwhitespacecollapse{true}%
269        \def\FOwrapoption{wrap}%
270        \def\FOtextalign{start}%
271        \def\FOfontfamily{#1}%
272        \def\FOfontsize{#6}%
273        \def\FOfontstretch{#2}%
274        \def\FOfontvariant{#4}%
```

```
        \def\FOfontweight{#5}%
        \def\FOfontstyle{#3}#8\global\FOinOutputfalse}}}%
}
```

We define here the action associated to `DummyRegion`, it is empty.

```
\expandafter\def\csname Static:DummyRegion\endcsname{}
\XMLNSAX{fo}{flow-name}{\FOflowname}{}
```

4.4 Flows

The content of the `<fo:flow>` formatting object is a sequence of blocks that defines a sequence of pages. The action associated is a bit complicated; for this reason, we shall define some pseudo functions instead of the original big code. Let's recall that the `<fo:page-sequence>` element remembers in `\pendingID` the current id and in `\PageNumber` the value of the initial page number. What we do here is to evaluate these instructions. We start with `\clearpage`, this makes sure that all pages are shipped out. In the case where the desired page number is 'auto' we are done; if it is auto-odd or auto-even, we emit an empty page, if needed, so as to make sure that the page number has the given parity. Question: why not emit a blank page? Finally, the page number can be an integer, in this case, we set the page counter. From now one, the page number is correct, so that we can insert the label of the parent. It is however too early to typeset something, for instance, `\textwidth` is still random.

```
\def\jg@flowI{%
  \clearpage
  \ifx\PageNumber\att@auto
  \else
     \ifx\PageNumber\att@autoeven
       \ifodd\c@page\hbox{}\newpage\fi
     \else
        \ifx\PageNumber\att@autoodd
          \ifodd\c@page\else\hbox{}\newpage\fi
        \else
          \setcounter{page}{\PageNumber}%
        \fi
     \fi
  \fi
  \let\FOid\pendingID \FOlabel % hacked jg
}
```

After evaluation of the children of the flow, we terminate with a `\clearpage`. We look at the force-page-count trait. If this imposes that the last page be even or odd, we may emit a blank page (note that the page counter holds one more than the last page number).

```
\def\jg@flowV{%
   \clearpage
   \ifx\ForcePage\att@auto
   \else
     \ifx\ForcePage\att@endoneven
       \ifodd\c@page\else\BlankPage\newpage\fi
     \else
        \ifx\ForcePage\att@endonodd
          \ifodd\c@page\BlankPage\newpage\fi
        \fi
     \fi
```

```
307     \fi
308 }
```

We define now four quantities `\PEven`, `\POdd`, `\PBlank` and `\PFirst` according to the master-reference; this can be the name of a page-sequence-master or a page-master. In the second case, we put this name into all four commands. Otherwise, the reference is something like 'twoside1' and the command `\Odd:twoside1` is a pagemaster. We put this value in `\POdd`. We do the same for the other commands. However, if `\Peven` is undefined, we use the value of `\POdd` instead. The case of `\PFirst` is a bit more complicated: in fact, if it is undefined, we look at the parity of the current page number. The code shown here is much simpler than the initial one, but has the same effects.

```
309 \def\jg@flowII{
310   \@ifundefined{Atomic:\CurrentPageMaster}
311    {
312     \edef\PFirst{\csname First:\CurrentPageMaster\endcsname}
313     \edef\PBlank{\csname Blank:\CurrentPageMaster\endcsname}
314     \edef\PEven{\csname Even:\CurrentPageMaster\endcsname}
315     \edef\POdd{\csname Odd:\CurrentPageMaster\endcsname}
316     \def\tmp{\relax}
317     \ifx\PBlank\tmp\let\PBlank\POdd\fi
318     \ifx\PEven\tmp\let\PEven\POdd\fi
319     \ifx\PFirst\tmp
320       \ifodd\c@page\let\PFirst\POdd \else\let\PFirst\PEven\fi\fi
321   }
322   {%
323       \let\PFirst\CurrentPageMaster
324       \let\PBlank\CurrentPageMaster
325       \let\POdd\CurrentPageMaster
326       \let\PEven\CurrentPageMaster
327   }
328 }
```

We use the `\PEven` command to define four commands, that contain the name of the header, footer, and their extent. The same is done for the other type of pages.

```
329 \def\jg@set@headings{
330    \edef\EvenHeadExtent{\csname\PEven:before-extent\endcsname}
331    \edef\EvenHead{Static:\csname\PEven:before\endcsname}
332    \edef\EvenTailExtent{\csname\PEven:after-extent\endcsname}
333    \edef\EvenTail{Static:\csname\PEven:after\endcsname}
334    \edef\FirstHeadExtent{\csname\PFirst:before-extent\endcsname}
335    \edef\FirstHead{Static:\csname\PFirst:before\endcsname}
336    \edef\FirstTailExtent{\csname\PFirst:after-extent\endcsname}
337    \edef\FirstTail{Static:\csname\PFirst:after\endcsname}
338    \edef\OddHeadExtent{\csname\POdd:before-extent\endcsname}
339    \edef\OddHead{Static:\csname\POdd:before\endcsname}
340    \edef\OddTailExtent{\csname\POdd:after-extent\endcsname}
341    \edef\OddTail{Static:\csname\POdd:after\endcsname}
342    \edef\BlankHeadExtent{\csname\PBlank:before-extent\endcsname}
343    \edef\BlankHead{Static:\csname\PBlank:before\endcsname}
344    \edef\BlankTailExtent{\csname\PBlank:after-extent\endcsname}
345    \edef\BlankTail{Static:\csname\PBlank:after\endcsname}
346 }
```

We define here a function that fetches some of these parameters. The value stored in `\themargin` will be defined later. It depends on the parity of the page number. This was not in the original fotex.sty.

```
\def\jg@use@blankpage{%
    \def\@thehead{\csname\BlankHead\endcsname}%
    \def\@thefoot{\csname\BlankTail\endcsname}%
     \ifodd\count\z@  \let\@themargin\oddsidemargin
     \else  \let\@themargin\evensidemargin\fi
    \def\headheight{\BlankHeadExtent}%
    \def\tailheight{\BlankTailExtent}}%
```

We consider here three commands that handle the other cases. It is assumed that the first page is always odd.

```
\def\jg@use@specialpage{%
    \def\@thehead{\csname\FirstHead\endcsname}%
    \def\@thefoot{\csname\FirstTail\endcsname}%
    \let\@themargin\oddsidemargin
    \def\headheight{\FirstHeadExtent}%
    \def\tailheight{\FirstTailExtent}}%
\def\jg@use@evenpage{%
    \def\@thehead{\csname\EvenHead\endcsname}%
    \def\@thefoot{\csname\EvenTail\endcsname}%
    \let\@themargin\evensidemargin
    \def\headheight{\EvenHeadExtent}%
    \def\tailheight{\EvenTailExtent}}%
\def\jg@use@oddpage{%
    \def\@thehead{\csname\OddHead\endcsname}%
    \def\@thefoot{\csname\OddTail\endcsname}%
    \let\@themargin\oddsidemargin
    \def\headheight{\OddHeadExtent}%
    \def\tailheight{\OddTailExtent}}%
```

This is how we select one of these four commands.

```
\newif\if@blankpage
\def\jg@usepagestyle{%
 \if@blankpage
    \jg@use@blankpage
 \else \if@specialpage
   \jg@use@specialpage
 \else \ifodd\count\z@
   \jg@use@oddpage\else \jg@use@evenpage
 \fi\fi\fi
 \global\@specialpagefalse
 \global\@blankpagefalse}
```

This defines a blank page. The original code was wrong. Here we use a new boolean.

```
\def\BlankPage{%
 \global\@blankpagetrue
 \mark{}%
 \hbox{}}
```

The original output routine contains: `\hb@xt@\textwidth{\@thehead}`. In the code that follows, we removed the line that decreases the text width by `\FOheadindent`, because this value is always zero. We also replace `\vfil` by `\vss`, in the case of a border.

```
\def\jg@headings#1#2{%
 \@tempdima\textwidth
      \setbox\@tempboxa \vbox to #1{%
        \color@hbox
          \normalcolor
         \hb@xt@\textwidth{\hfill\llap{\hb@xt@\@tempdima{#2}}}%
        \color@endbox
        \vss%  \vfil
        }%
      \dp\@tempboxa \z@
      \box\@tempboxa}%
```

This is the modified output routine.

```
\def\@outputpage{%
\begingroup           % the \endgroup is put in by \aftergroup
  \let \protect \noexpand
%  \@resetactivechars
  \@parboxrestore
  \shipout \vbox{%
    \set@typeset@protect
    \aftergroup \endgroup
    \aftergroup \set@typeset@protect
  \jg@usepagestyle
  \reset@font
  \normalsize
  \normalsfcodes
  \let\label\@gobble
  \let\index\@gobble
  \let\glossary\@gobble
  \baselineskip\z@skip \lineskip\z@skip \lineskiplimit\z@
   \@begindvi
    \vskip \topmargin
    \moveright\@themargin \vbox {%
      \jg@headings{\headheight}{\@thehead}%
      \vskip \headsep
      \box\@outputbox
      \baselineskip \footskip
      \vskip \bottommargin
      \jg@headings{\tailheight}{\@thefoot}%
      }%
    }%
  \global \@colht \textheight
  \stepcounter{page}%
  \let\firstmark\botmark
}
```

Bootstrap code. Seems useless. As a consequence, no variable is added for blank pages.

```
\gdef\OddTail {}
\gdef\OddHead {}
\gdef\EvenTail {}
```

```
\gdef\EvenHead {}
\gdef\FirstTail {}
\gdef\FirstHead {}
\gdef\OddTailExtent{\z@}
\gdef\OddHeadExtent{\z@}
\gdef\EvenTailExtent{\z@}
\gdef\EvenHeadExtent{\z@}
\gdef\FirstTailExtent{\z@}
\gdef\FirstHeadExtent{\z@}
```

We simplified the code by removing all references to a quantity SpecialOffset that was always zero. We have already explained what the `\Pass` command does: it fetches some parameters from the region-body (note the fixed name here); they are `\columnsep`, `\NColumns`, `\Marginbottom` and `\Margintop`. We also evaluate `\Atomic:XXX`. The result is to define `\MasterXXXMargins`, as well as `\paperwidth` and `\paperheight`. We hope that both evaluations yield the same result. The only difference between them is that the left margin can change (we hope that the sum of the left and right margins are the same). All these variables will be used by `\FOSetPage`, and forgotten after that.

```
\def\jg@flowIII{%
  \expandafter\Pass\csname\POdd:xsl-region-body\endcsname\\
  \csname Atomic:\POdd\endcsname
  \oddsidemargin\MasterLeftMargin
  \csname Atomic:\PEven\endcsname
  \evensidemargin\MasterLeftMargin
  \jg@set@headings
  \FOSetPage}
```

The purpose of this command is to compute the text height and text width, and to store it wherever needed (`\hsize`, `\vsize`, `\@colht`, etc). We also remember three quantities: `\bottommargin`, `\headsep` and `\topmargin`.

```
\def\FOSetPage{%
 \bottommargin\Marginbottom
 \headsep\Margintop
 \topmargin\MasterTopMargin
 \textheight\paperheight
 \textwidth\paperwidth
 \advance\textheight by -\FirstHeadExtent
 \advance\textheight by -\FirstTailExtent
 \advance\textheight by -\MasterTopMargin
 \advance\textheight by -\Margintop
 \advance\textheight by -\MasterBottomMargin
 \advance\textheight by -\Marginbottom
 \advance\textwidth by -\MasterLeftMargin
 \advance\textwidth by -\MasterRightMargin
 \FOpdfsetpagesize{\paperwidth}{\paperheight}
 \@colht\textheight
 \@colroom\textheight
 \vsize\textheight
 \linewidth\textwidth
 \columnwidth\textwidth
 \hsize\columnwidth \linewidth\hsize
 \def\headheight{12pt}%
 \global\@specialpagetrue}
```

This might be useful.

```
\def\FOpdfsetpagesize#1#2{%
 \@ifundefined{pdfoutput}{}{%
   \global\pdfpagewidth\paperwidth
   \global\pdfpageheight\paperheight}}
```

This is the flow element. With all these simplifications and auxiliary commands, the code becomes easy to understand.

```
\XMLelement{fo:flow}
  {}
  {\FOSetHyphenation
   \jg@flowI
   \jg@flowII
   \jg@flowIII
    \ifnum\NColumns>1\begin{multicols}{\NColumns}\fi
    }
  {
   \ifnum\NColumns>1\end{multicols}\fi
   \jg@flowV
   }
```

4.5 Borders

A border has three properties: a color, a style and a width. There are four borders. They are called 'before', 'after', 'start' and 'end'. These quantities are called relative, because they depend on the writing-mode trait. We assume that this is 'lr-td' (Inline components and text within a line are written left-to-right. Lines and blocks are placed top-to-bottom). In such a case, 'before' is 'top', 'after' is 'bottom', 'start' is 'left' and 'end' is 'right'. Quantities like 'top' and 'bottom' are called absolute. We shall use absolute quantities only to set relative quantities.

Here we declare the relative attributes.

```
\XMLNSA{fo}{border-before-color}{\FOborderbeforecolor}{\FOcolor}
\XMLNSA{fo}{border-after-color}{\FOborderaftercolor}{\FOcolor}
\XMLNSA{fo}{border-start-color}{\FOborderstartcolor}{\FOcolor}
\XMLNSA{fo}{border-end-color}{\FOborderendcolor}{\FOcolor}

\XMLNSAX{fo}{border-before-style}{\FOborderbeforestyle}{none}
\XMLNSAX{fo}{border-after-style}{\FOborderafterstyle}{none}
\XMLNSAX{fo}{border-start-style}{\FOborderstartstyle}{none}
\XMLNSAX{fo}{border-end-style}{\FOborderendstyle}{none}

\XMLNSA{fo}{border-before-width}{\FOborderbeforewidth}{medium}
\XMLNSA{fo}{border-after-width}{\FOborderafterwidth}{medium}
\XMLNSAX{fo}{border-start-width}{\FOborderstartwidth}{medium}
\XMLNSAX{fo}{border-end-width}{\FOborderendwidth}{medium}
```

Here we declare the absolute attributes. The default value is a special marker.

```
\XMLNSA{fo}{border-top-color}{\FObordertopcolor}{\LINK}
\XMLNSA{fo}{border-bottom-color}{\FOborderbottomcolor}{\LINK}
\XMLNSA{fo}{border-left-color}{\FOborderleftcolor}{\LINK}
\XMLNSA{fo}{border-right-color}{\FOborderrightcolor}{\LINK}

```

```
\XMLNSAX{fo}{border-top-style}{\FObordertopstyle}{\LINK}
\XMLNSAX{fo}{border-bottom-style}{\FOborderbottomstyle}{\LINK}
\XMLNSAX{fo}{border-left-style}{\FOborderleftstyle}{\LINK}
\XMLNSAX{fo}{border-right-style}{\FOborderrightstyle}{\LINK}

\XMLNSAX{fo}{border-top-width}{\FObordertopwidth}{\LINK}
\XMLNSAX{fo}{border-bottom-width}{\FOborderbottomwidth}{\LINK}
\XMLNSAX{fo}{border-left-width}{\FOborderleftwidth}{\LINK}
\XMLNSAX{fo}{border-right-width}{\FOborderrightwidth}{\LINK}

\XMLNSAX{fo}{border-left}{\FOborderleft}{\LINK}
\XMLNSAX{fo}{border-right}{\FOborderright}{\LINK}
\XMLNSAX{fo}{border-top}{\FObordertop}{\LINK}
\XMLNSAX{fo}{border-bottom}{\FOborderbottom}{\LINK}
```

The attribute border is a shorthand for all four borders. Its value is a list of three items: width, style and color. This command sets everything.

```
\def\expandBorder#1 #2 #3\\{%
    \def\FOborderstartcolor{#3}%
    \def\FOborderendcolor{#3}%
    \def\FOborderbeforecolor{#3}%
    \def\FOborderaftercolor{#3}%
    \def\FOborderstartwidth{#1}%
    \def\FOborderendwidth{#1}%
    \def\FOborderbeforewidth{#1}%
    \def\FOborderafterwidth{#1}%
    \def\FOborderstartstyle{#2}%
    \def\FOborderendstyle{#2}%
    \def\FOborderbeforestyle{#2}%
    \def\FOborderafterstyle{#2}}
```

We declare here the border attribute. There are three other attributes that specify only one quantity (color, width, style) for all four borders.

```
\XMLNSAX{fo}{border}{\FOborder}{}
\XMLNSA{fo}{border-color}{\FObordercolor}{black}
\XMLNSAX{fo}{border-width}{\FOborderwidth}{}
\XMLNSAX{fo}{border-style}{\FOborderstyle}{}
```

We declare here the padding variables[6].

```
\XMLNSAX{fo}{padding}{\FOpadding}{\z@}

\XMLNSAX{fo}{padding-top}{\FOpaddingtop}{\z@}
\XMLNSAX{fo}{padding-bottom}{\FOpaddingbottom}{\z@}
\XMLNSAX{fo}{padding-left}{\FOpaddingleft}{\z@}
\XMLNSAX{fo}{padding-right}{\FOpaddingright}{\z@}

\XMLNSAX{fo}{padding-before}{\FOpaddingbefore}{\z@}
\XMLNSAX{fo}{padding-after}{\FOpaddingafter}{\z@}
\XMLNSAX{fo}{padding-start}{\FOpaddingstart}{\z@}
\XMLNSAX{fo}{padding-end}{\FOpaddingend}{\z@}
```

Here we declare the attributes for the margins, and we define default values. These values are absolute. The corresponding relative properties are space-before, space-after, space-start, and

[6] It seems that fotex does not use absolute values.

space-end. The quantities space-before and space-after are defined for block level formatting objects (in `\SpaceAttributes`), while space-start, and space-end are for inline objects[7]. There are also two quantities start-indent, end-indent which are related to margins, but are a bit complicated. They are defined later.

```
\XMLNSAX{fo}{margin}{\FOmargin}{}
\XMLNSAX{fo}{margin-left}  {\FOmarginleft}  {0pt}
\XMLNSAX{fo}{margin-right}  {\FOmarginright}  {0pt}
\XMLNSAX{fo}{margin-top}  {\FOmargintop}  {0pt}
\XMLNSAX{fo}{margin-bottom}  {\FOmarginbottom}  {0pt}

\gdef\FOmarginbottom{\z@}
\gdef\FOmarginleft{\z@}
\gdef\FOmarginright{\z@}
\gdef\FOmargintop{}
```

We could simplify a bit the code of the next command, using the pseudo command \eval-thin-med-thick. This command sets the width to zero in case the style is not 'solid'; otherwise, it sets the width to 0.4pt, 0.8pt or 1.2pt if it is 'thin', 'medium' or 'thick'. It sets the boolean `\ifFOBlockGrab` to true. We could also use the pseudo command `\set4{X}{Y}`: if border-X is Y, is does nothing, otherwise it sets border-Z-X to border-X. Here Z is one of 'before', 'after', 'start', or 'end'. Note: Absolute values have precedence over relative values, hence the 12 first lines of the code. The line marked 'JG' was not in the original...

```
\def\FOexpandattributes{%
\ifx\FObordertopstyle\LINK\else\let\FOborderbeforestyle\FObordertopstyle\fi
\ifx\FOborderbottomstyle\LINK\else\let\FOborderafterstyle\FOborderbottomstyle\fi
\ifx\FOborderrightstyle\LINK\else\let\FOborderendstyle\FOborderrightstyle\fi
\ifx\FOborderleftstyle\LINK\else\let\FOborderstartstyle\FOborderleftstyle\fi
\ifx\FObordertopwidth\LINK\else\let\FOborderbeforewidth\FObordertopwidth\fi
\ifx\FOborderbottomwidth\LINK\else\let\FOborderafterwidth\FOborderbottomwidth\fi
\ifx\FOborderrightwidth\LINK\else\let\FOborderendwidth\FOborderrightwidth\fi
\ifx\FOborderleftwidth\LINK\else\let\FOborderstartwidth\FOborderleftwidth\fi
\ifx\FObordertopcolor\LINK\else\let\FOborderbeforecolor\FObordertopcolor\fi
\ifx\FOborderbottomcolor\LINK\else\let\FOborderaftercolor\FOborderbottomcolor\fi
\ifx\FOborderrightcolor\LINK\else\let\FOborderendcolor\FOborderrightcolor\fi
\ifx\FOborderleftcolor\LINK\else\let\FOborderstartcolor\FOborderleftcolor\fi
  \ifx\FObordercolor\att@black
  \else
    \let\FOborderstartcolor\FObordercolor
    \let\FOborderendcolor\FObordercolor
    \let\FOborderbeforecolor\FObordercolor
    \let\FOborderaftercolor\FObordercolor
  \fi
  \ifx\FOborderwidth\@empty
  \else
    \let\FOborderstartwidth\FOborderwidth
    \let\FOborderendwidth\FOborderwidth
    \let\FOborderbeforewidth\FOborderwidth
    \let\FOborderafterwidth\FOborderwidth
  \fi
  \ifx\FOborderstyle\@empty
  \else
```

[7] They are unused by fotex.

```
    \let\FOborderstartstyle\FOborderstyle
    \let\FOborderendstyle\FOborderstyle
    \let\FOborderbeforestyle\FOborderstyle
    \let\FOborderafterstyle\FOborderstyle
  \fi
  \ifx\FOborder\@empty
  \else
    \expandafter\expandBorder\FOborder\\{}%
  \fi
  \ifdim\FOpadding>\z@
    \let\FOpaddingstart\FOpadding
    \let\FOpaddingend\FOpadding
    \let\FOpaddingbefore\FOpadding
    \let\FOpaddingafter\FOpadding
  \fi
  \ifx\FOmargin\@empty
  \else
    \let\tmpmargin\FOmargin
    \let\FOmarginleft\tmpmargin
    \let\FOmarginright\tmpmargin
    \let\FOmargintop\tmpmargin
    \let\FOmarginbottom\tmpmargin
  \fi
  \ifx\FOborderendstyle\att@solid
   \ifx\FOborderendwidth\att@thin\def\FOborderendwidth{0.4pt}\fi
   \ifx\FOborderendwidth\att@medium\def\FOborderendwidth{0.8pt}\fi
   \ifx\FOborderendwidth\att@thick\def\FOborderendwidth{1.2pt}\fi
   \FOBlockGrabtrue
  \else
   \def\FOborderendwidth{\z@}%
  \fi
  \ifx\FOborderstartstyle\att@solid
   \ifx\FOborderstartwidth\att@thin\def\FOborderstartwidth{0.4pt}\fi
   \ifx\FOborderstartwidth\att@medium\def\FOborderstartwidth{0.8pt}\fi
   \ifx\FOborderstartwidth\att@thick\def\FOborderstartwidth{1.2pt}\fi
   \FOBlockGrabtrue
  \else
   \def\FOborderstartwidth{\z@}%
  \fi
  \ifx\FOborderafterstyle\att@solid
   \ifx\FOborderafterwidth\att@thin\def\FOborderafterwidth{0.4pt}\fi
   \ifx\FOborderafterwidth\att@medium\def\FOborderafterwidth{0.8pt}\fi
   \ifx\FOborderafterwidth\att@thick\def\FOborderafterwidth{1.2pt}\fi
   \FOBlockGrabtrue %%  <---  JG
  \else
   \def\FOborderafterwidth{\z@}%
  \fi
  \ifx\FOborderbeforestyle\att@solid
   \ifx\FOborderbeforewidth\att@thin\def\FOborderbeforewidth{0.4pt}\fi
   \ifx\FOborderbeforewidth\att@medium\def\FOborderbeforewidth{0.8pt}\fi
   \ifx\FOborderbeforewidth\att@thick\def\FOborderbeforewidth{1.2pt}\fi
   \FOBlockGrabtrue
```

```
641   \else
642    \def\FOborderbeforewidth{\z@}%
643   \fi}
```

4.6 Spacing for blocks

In XSL/Format a block is the equivalent of a TEX box. It can define some space before it, and after it. If we have two blocks, one with some space x after it and another one, with some space y before it, there are some precedence rules that explain what to do. These are not implemented in fotex. However assume that we have a sequence of spaces; each such sequence is defined by a triple (x_1, x_2, x_3), optimum, maximum, and minimum value. If I understand correctly, the following happens. First, the largest sequence of spaces is considered. Then, conditionality is considered; this is used at the start or end of a page, at the start or end of a line (in LATEX, it is the difference between `\hspace` and `\hspace*`); in such a case, all initial conditional spaces are removed. If any of these spaces is forcing, the result is the sum of the forcing spaces, otherwise, the merge of them. When merging spaces, only those of highest priority are used. Consider two of them (x_1, x_2, x_3), and (y_1, y_2, y_3). If $x_1 \neq y_1$ (different optimum values), then the one with lowest optimum value is discarded. Otherwise, the result is $(x_1 = y_1, \min(x_2, y_2), \max(x_3, y_3))$.

These spaces are local to a block, hence are not globally defined.

```
644 \def\SpaceAttributes{
645  \XMLattributeX{space-after.optimum}{\FOspaceafteroptimum}{\z@}
646  \XMLattributeX{space-after.maximum}{\FOspaceaftermaximum}{\z@}
647  \XMLattributeX{space-after.minimum}{\FOspaceafterminimum}{\z@}
648  \XMLattributeX{space-before.optimum}{\FOspacebeforeoptimum}{\z@}
649  \XMLattributeX{space-before.maximum}{\FOspacebeforemaximum}{\z@}
650  \XMLattributeX{space-before.minimum}{\FOspacebeforeminimum}{\z@}
651  \XMLattributeX{space-after}{\FOspaceafter}{}
652  \XMLattributeX{space-before}{\FOspacebefore}{}}
```

This is how we use these space-after attributes. Original code was inlined. Let A, B, and C, be the optimum, minimum and maximum values. The specifications say that, if `\FOspaceafter` is given, this should be the value of non-provided A, B and C. It also states that, if margin-top is defined, then setting space-before.minimum will have no effect. The code that follows does not implement these subtleties. Assume that the values are 2, 3 and 4pt. Then we could use some glue of value `3pt plus 1pt minus 1pt`. The shrink part is $A-B$, the stretch part is $C-A$. This code uses $A+C$ instead (strange). In fact, the stretch value can grow arbitrarily in TEX; said otherwise, we cannot implement exactly the XSL/Format mechanism. The argument of this command is a skip register, that contains the desired glue, or a TEX command that uses the glue.

```
653 \def\jg@usespaceafter#1{
654   \ifx\@empty\FOspaceafter
655       \@tempdima\FOspaceafteroptimum
656       \advance\@tempdima by -\FOspaceafterminimum
657       \@tempdimb\FOspaceafteroptimum
658       \advance\@tempdimb by \FOspaceaftermaximum
659       #1\FOspaceafteroptimum plus \@tempdimb minus \@tempdima
660     \else
661       #1\FOspaceafter
662     \fi}
```

This is how we use these space-before attributes. Original code was inlined. The code is as above.

```
\def\jg@usespacebefore#1{
    \ifx\@empty\FOspacebefore
      \@tempdima\FOspacebeforeoptimum
      \advance\@tempdima by -\FOspacebeforeminimum
      \@tempdimb\FOspacebeforeoptimum
      \advance\@tempdimb by \FOspacebeforemaximum
      #1\FOspacebeforeoptimum plus \@tempdimb minus \@tempdima
    \else
      #1\FOspacebefore
    \fi}
```

Attributes for keep-together. There are three sub-cases: within line, page, column. The value can be 'always', 'auto', or a number. It corresponds in TeX to some penalty: 0 for 'auto', 10000 for 'always'. Otherwise, the number should produce something between these two values (currently ignored). Keep-within-line means a penalty in horizontal mode, otherwise in vertical mode. Note that it is not possible to associate a penalty to a column-break (in TeX switching from one column to the other is the same as switching from one page to the other; the difference is how `\output` handles these cases). The within-line case is not implemented.

```
\XMLNSAX{fo}{keep-together}{\FOkeeptogether}{\inherit}
\XMLNSAX{fo}{keep-together.within-column} {\FOkeeptogetherColumn}{\inherit}
\XMLNSAX{fo}{keep-together.within-page} {\FOkeeptogetherPage}{\inherit}
\XMLstringX\att@always<>always</>
```

Attributes for keep-with-next. As above. There is also a keep-with-previous, but this is not implemented. Too bad.

```
\XMLNSAX{fo}{keep-with-next}{\FOkeepwithnext}{auto}
\XMLNSAX{fo}{keep-with-next.within-column}{\FOkeepwithnextColumn}{auto}
\XMLNSAX{fo}{keep-with-next.within-page} {\FOkeepwithnextPage}{auto}
```

This was inlined. We call `\samepage` if no pagebreak should occur.

```
\def\jg@keep@together{%
  \ifx\FOkeeptogether\att@always\samepage\fi
  \ifx\FOkeeptogetherColumn\att@always\samepage\fi
  \ifx\FOkeeptogetherPage\att@always\samepage\fi}
```

This is a bit more complicated: we set a switch that says that we have to keep this item with the next one.

```
\def\jg@keepnext{%
   \@tempswafalse
   \ifx\FOkeepwithnext\att@always\@tempswatrue\fi
   \ifx\FOkeepwithnextColumn\att@always\@tempswatrue\fi
   \ifx\FOkeepwithnextPage\att@always\@tempswatrue\fi}
```

In order to understand the following code, you must know that there are four kinds of blocks. If `\ifFOinOutput` is true, we are typesetting a static area (page headers and footers). If `\FOinTable` is positive, we are in a table, and a special case is when we typeset the label of an item in a list. Originally, the code did some action if `\FOTableNesting` was positive; however, it is currently impossible to nest tables, and the counter is never modified.

This inserts some vertical space. We compute in `\@tempskipa` the quantity to add and in `\@tempswa` a boolean value that says whether or not a penalty should be inserted. We insert the penalty and the skip. This resets to zero the value of `\FOspacebefore`.

```
\def\FOvspacebefore{%
  \ifFOinOutput
  \else
```

```
692       \jg@usespacebefore{\@tempskipa}
693       \jg@keepnext
694       \if@tempswa\addpenalty\@secpenalty\fi
695       \addvspace\@tempskipa
696     \fi
697     \def\FOspacebefore{\z@}}
```

This adds some vertical space after a block. Nothing is done in table headings. Note that we use an infinite penalty here. I'm not sure this is a good idea.

```
698 \newskip\FOafterskip
699 \def\FOvspaceafter{%
700   \ifFOinOutput
701   \else
702    \jg@usespaceafter{\FOafterskip}
703    \jg@keepnext
704    \if@tempswa\addpenalty{\@M}\fi
705    \addvspace\FOafterskip
706   \fi}
```

4.7 Quadding

The value of the text-align can be one of the following: start, end, center, justify, inside, outside, left, right, or a string. The value of text-align-last can be any of these, plus relative. The value 'relative' means that forced lines behave like other ones, except if the text is justified, case where forced lines are left aligned (in TEX, this corresponds to a default `\parfillskip`). By forced line, we mean either the last in a paragraph, or one induced by evaluation of the character U+000A (this is not implemented in fotex; note however that U+2028 calls `\newline`). This is a CSS property, adapted to XSL/Format; for this reason 'left' is the same as 'start' and 'right' as 'end'. If alignment is start, it means that there is no space between the first character and the margin, if it is end, then there is no space between the last character and the margin.

We declare the attributes.

```
707 \XMLNSAX{fo}{text-align}{\FOtextalign}{\inherit}
708 \XMLNSAX{fo}{text-align-last}{\FOtextalignlast}{\inherit}
709 \XMLstringX\att@relative<>relative</>
710 \gdef\FOtextalign{start}
711 \gdef\FOtextalignlast{relative}
```

The indentation (left and right) is given by two quantities, stored in `\FOstartindent` and `\FOendindent`. In the case where we have a list and an item in a list, the start of the body is defined by `body-start()` and the end of the label by `label-end()`. In fact, there are two traits, one that indicates the distance between the starts of the label and the body, and one that indicates the distance between the end of the label and the start of the body.

```
712 \XMLNSA{fo}{start-indent}{\FOstartindent}{\inherit}
713 \XMLNSA{fo}{end-indent}{\FOendindent}{\inherit}
714 \XMLstring\att@labelend<>label-end()</>
715 \XMLstring\att@bodystart<>body-start()</>
```

We use here two commands that return zero in the case where the attribute value is one of the functions mentioned above, and a third one that set these quantities to zero.

```
716 \gdef\EndIndent{\ifx\FOendindent\att@labelend\z@\else\FOendindent\fi}
717 \gdef\StartIndent{\ifx\FOstartindent\att@bodystart\z@\else\FOstartindent\fi}
718 \def\jg@hackindent{
```

```
719    \ifx\FOstartindent\att@bodystart \let\FOstartindent\z@ \fi
720    \ifx\FOendindent\att@labelend \let\FOendindent\z@ \fi}
```

We use three commands `\QuaddingStart`, `\Quadding` and `\QuaddingEnd`. The first function is defined as follows. This does not seem to correspond to the explanations given above.

```
721 \def\QuaddingStart{%
722  \ifx\FOtextalignlast\att@relative
723    \csname startQ@\FOtextalign\endcsname
724  \else
725    \csname startQ@\FOtextalignlast\endcsname
726  \fi}
```

Function two for quadding.

```
727 \def\QuaddingEnd{%
728  \ifx\FOtextalignlast\att@relative
729    \csname endQ@\FOtextalign\endcsname
730  \else
731    \csname endQ@\FOtextalignlast\endcsname
732  \fi}
```

Function three for quadding.

```
733 \def\Quadding{%
734  \ifx\FOtextalignlast\att@relative
735    \csname Q@\FOtextalign\endcsname
736  \else
737    \csname Q@\FOtextalignlast\endcsname
738  \fi}
```

Remaining code in this paragraph comes from the file mlnames.sty. These commands describe the action in the case where the text should be centered. Note that line separator character U+2028 is bound to `\newline`, and needs to be redefined. Why is `\Q@centered` defined? The code of `\Q@center` is interesting. Assume that start- and end-indent are a and b respectively. In order to center the text in a region where a have been removed on the left and b on the right, we can put a plus 1fil in `\leftskip`, b plus 1fil in `\rightskip`. In fact we subtract $a + b$ from both these quantities (this does not change the alignment). Question: why do we change `\@rightskip`?

```
739 \def\startQ@center{\hskip\z@ plus 1filll}
740 \def\endQ@center{\hskip\z@ plus 1filll}
741 \def\Q@center{%
742   \let\newline\@centercr
743   \rightskip-\StartIndent plus 1fil%
744   \@rightskip\rightskip
745   \leftskip-\EndIndent plus 1fil%
746   \parfillskip\z@skip
747 }
748 \let\Q@centered\Q@center
```

This is in the case right justified, case where alignment is `right` or `end`.

```
749 \def\startQ@end{\hfill}
750 \def\endQ@end{}
751 \def\Q@end{
752   \let\newline\@centercr
753   \leftskip\StartIndent plus 1fil%
754   \parfillskip\z@skip
755 }
```

```
\let\startQ@right\startQ@end
\let\endQ@right\endQ@end
\let\Q@right\Q@end
```

This is in the case left justified, case where alignment is `start` or `left`.

```
\def\startQ@start{}
\def\endQ@start{\hfill}
\def\Q@start{
  \let\newline\@centercr
  \rightskip\EndIndent plus 1fil
  \@rightskip\rightskip
  \leftskip\StartIndent
  \parfillskip\z@skip
}
\let\startQ@left\startQ@start
\let\endQ@left\endQ@start
\let\Q@left\Q@start
```

This is in the case justified. Why do we need a definition for `justified` ?

```
\def\startQ@justify{}
\def\endQ@justify{}

\def\startQ@justified{%
  \leftskip\StartIndent
  \rightskip\EndIndent
  \@rightskip\rightskip
}
\def\Q@justified{%
  \parfillskip\@flushglue
  \leftskip\StartIndent
  \rightskip\EndIndent
  \@rightskip\rightskip
}
\def\endQ@justified{}
\let\Q@justify\Q@justified
```

Case empty. Strange.

```
\let\startQ@\startQ@justified
\let\endQ@\endQ@justified
\let\Q@\Q@justified
```

This is what the documentation says if the value of text-align is `inside`: If the page binding edge is on the start-edge, the alignment will be `start`. If the binding is the end-edge, the alignment will be `end`. If neither, use `start` alignment. For `outside`, it is the opposite. If I understand correctly, this may depend on the parity of the page, hence cannot be implemented in TEX.

```
\def\startQ@pageoutside{\hfill}
\def\endQ@pageoutside{}

\def\startQ@pageinside{}
\def\endQ@pageinside{\hfill}
```

4.8 Arrays

Implementing arrays is a bit complicated. It uses a lot of variables. There were also different tentatives, so that some variables and tests have no usage anymore.

We start with a piece of code that remembers the column widths. This declares a counter.

```
\newcount\arraylength
```

After `\Array{foo}[bar]{gee}`, the command `\foobar` contains `gee`. This is assumed to be the width of column 'bar' of array 'foo'.

```
\def\Array#1[#2]#3{%
  \expandafter\xdef\csname #1#2\endcsname{#3}}
```

Initialization of the `foo` array. Column 0 of the array is set to empty. In the original code, this constructed `\foo` so that `\foo[bar]` calls `\foobar`, but the command was never used.

```
\def\DeclareArray#1{%
   \Array{#1}[0]{}}
```

This finds the length of an array by considering the first `\csname` that produces `\relax` as result (i.e. first undefined slot).

```
\def\getArraylength#1{%
   \arraylength0
   \loop\expandafter\ifx\csname #1\the\arraylength\endcsname\relax%
   \else\advance\arraylength by1\repeat}%
```

We find the end of the array, then insert something there.

```
\def\addToArray#1#2{\getArraylength{#1}%
        \Array{#1}[\the\arraylength]{#2}}%
```

This removes from memory everything associated to this table. Since it is not possible to remove the command from the hash table, we set it to `\relax` (not undefined!).

```
\def\clearArray#1{\getArraylength{#1}%
        \loop\ifnum\arraylength >0%
        \global\expandafter\let\csname #1\the\arraylength\endcsname\relax%
        \advance\arraylength by-1\repeat}%
```

These are the variables used below.

```
\newcount\AbsoluteTableCount   % unique Id for a table
\newcount\CellCount            % index of a cell in a row
\newcount\FOinTable            % >0 if in a table
\newcount\NCols                % non-zero if col specs given
\newdimen\CurrentCellWidth     % width of current cell
\newdimen\TableWidth           % width of the table
\newif\ifFOFirstCell           % unused...
\def\TableHeader{}             % current table header
\NCols0
\FOinTable0
% \newcount\RowCount           % unused
% \newtoks\ColSpecs            % unused
```

This sets the width of the table to be the argument, minus the `\tabcolsep`, the left margin and the right margin.

```
\def\jg@settablewidth#1{%
   \TableWidth#1%
   \advance\TableWidth by -\tabcolsep
```

```
825     \advance\TableWidth by -\FOmarginleft
826     \advance\TableWidth by -\FOmarginright}
```

Same code, without the `\tabcolsep`.

```
827 \def\jg@settablewidth@alt#1{%
828     \TableWidth#1%
829     \advance\TableWidth by -\FOmarginleft
830     \advance\TableWidth by -\FOmarginright}
```

This initializes some other quantities.

```
831 \def\jg@tablesetup{%
832     \NCols0
833     \gdef\TableHeader{}%
834     \NoTableSetup}
```

This piece of code is executed at the start of a table or tabular. In the case of a tabular in a table, it will be executed twice. We commented out the code that increments the table counter: this is because it is currently impossible to put tables in tables; and once a table is typeset, all information about it is discarded. However, some names remain in the hash table.

```
835 \def\NoTableSetup{%
836   \ifx\FOwidth\att@auto\else     %% this test added by JG
837     \jg@settablewidth{\FOwidth}%
838  \fi
839   % \global\advance\AbsoluteTableCount by 1 %
840   \DeclareArray{fotable\the\AbsoluteTableCount:}%
841   \global\CellCount0 }
```

We declare here an attribute for the placement of tables. This is in the fotex namespace. We declare also the reference orientation. This can be used in an inline container, for turning things.

```
842 \XMLNSAX{fo}{fotex:placement}{\FOkplacement}{}
843 \XMLNSAX{fo}{reference-orientation}{\FOreferenceorientation}{0}
844 \XMLname{fo:inline-container}{\FOInlineContainer}
845 \gdef\FOkplacement{}
```

A table is defined by the `<fo:table-and-caption>` element, has `<fo:table-caption>` (optional) and `<fo:table>` (required) as children.

In the case of table-and-caption, we use a floating environment. This can be a table or a sideways table. We must close the same environment at the end.

```
846 \XMLelement{fo:table-and-caption}
847   {}
848   {
849    \jg@settablewidth{\linewidth}
850    \jg@tablesetup
851     \ifx\XML@parent\FOInlineContainer
852       \ifnum\FOreferenceorientation=0
853       \else \begin{sidewaystable}\fi
854     \else
855      \ifnum\FOreferenceorientation=0
856        \ifx\FOkplacement\@empty
857         \begin{table}[!htbp]\FOlabel
858          \else \edef\ktable{\noexpand\begin{table}[\FOkplacement]} \ktable \fi
859       \else \begin{sidewaystable}\fi
860     \fi
861     \FOlabel
```

```
  }
  {\ifx\XML@parent\FOInlineContainer
     \ifnum\FOreferenceorientation=0  \else \end{sidewaystable}  \fi
   \else
     \ifnum\FOreferenceorientation=0 \end{table} \else \end{sidewaystable} \fi
   \fi
  }
```

Typesetting the caption is trivial. In particular, we do not call `\caption`. The table has a caption-side trait that explains where to put the caption. This will be ignored.

```
\XMLelement{fo:table-caption}
  {}
  {}
  {\par}
```

A `<fo:table>` can be used inside or outside of a table-and-caption. The content: some `<fo:table-column>` elements, an optional `<fo:table-header>`, an optional `<fo:table-footer>` and some `<fo:table-body>` elements. Translation is trivial.

This is what we do for a table (equivalent of a LaTeX `tabular`). It has no header.

```
\XMLelement{fo:table}
  {}
  {
   \jg@settablewidth@alt{\linewidth}
   \jg@tablesetup
  }
  {}
```

The header of a table is not typeset now, but later. The footer is currently ignored. Too bad. The global definition here is one that forbids putting tables in tables.

```
\XMLelement{fo:table-header}
  {}
  {\xmlgrab}
  {\gdef\TableHeader{#1}}
```

This command is used to specify the width and other properties of one or more columns. The doc says: The number-columns-repeated property specifies the repetition of a `<fo:table-column>` specification n times; with the same effect as if the `<fo:table-column>` formatting object had been repeated n times in the result tree. The column-number property, for all but the first, is the column-number of the previous one plus the value of the number-columns-spanned property.

```
\XMLelement{fo:table-column}
 {}
 {
  \@tempcnta0
  \loop\ifnum\FOnumbercolumnsrepeated>\@tempcnta
   \advance\@tempcnta by 1
   {\NoTableColumn}%
  \repeat
  }
 {}
```

There is something wrong in this procedure: the column number is unused (said otherwise, if specifications are not in the order 1, 2, 3, etc, they will be stored in random order). Note: the column-number should not be zero. If at least one table column has been given, then `\Ncols` is not zero. The test to proportional-column-width is strange: what if the argument is not one? If

the value is a percentage, it refers to the width of the table. The computed value is stored in `\@tempdima`, and then in the array data structure.

```
\def\NoTableColumn{%
   \ifx\@empty\FOcolumnnumber
    \global\advance\NCols by 1
   \else
    \global\NCols\FOcolumnnumber
   \fi
   \ifx\prop@width\FOcolumnwidth\def\FOcolumnwidth{1in}\fi
   \ifx\@empty\FOcolumnwidth\def\FOcolumnwidth{1in}\fi
   \TablePercentToDimen{\FOcolumnwidth}%
   \addToArray{fotable\the\AbsoluteTableCount:}{\the\@tempdima}%
}
\XMLstringX\prop@width<>proportional-column-width(1)</>
```

The table body, header and footer contain either rows, or cells. Footers are currently ignored. Inside a table `\FOinTable` is 1.

```
\XMLelement{fo:table-body}
  {  }
  { \FOFirstCelltrue
    \FOinTable1
    \expandafter\NoTableStart{\TableHeader}%
  }
  { \NoTableEnd }
```

Action is trivial.

```
\def\NoTableStart#1{#1}
\def\NoTableEnd{\clearArray{fotable\the\AbsoluteTableCount:}}
```

We use a command for typesetting a row.

```
\XMLelement{fo:table-row}
  {}
  {\xmlgrab}
  {\NoTableRow{#1}}
```

We use two passes for our table rows. For the first pass the height of the row may be unknown; hence we use a boolean that says that it is the first or the second pass.

```
\newdimen\NoTableCellHeight
\newif\ifNoTableCheckHeight
```

This is for the first pass. We use a `\strut`, a capital letter and a letter with a descender. Note: it is two small if the cell has a border or padding.

```
\def\jg@default@cell@height{%
 \setbox0=\vbox{\strut They}%
 \NoTableCellHeight=\ht0
 \advance\NoTableCellHeight by \dp0
 \NoTableCheckHeightfalse}
```

In the first pass, we put the row in a box, and we compute the total height plus depth of the box. Then we look at page parameters to see if there is enough place on the current page; we may call `\clearpage`.

```
\def\jg@tablerow@firstpass#1{%
 \setbox0=\hbox{#1}%
 \@tempdima=\ht0
 \advance\@tempdima by \dp0
```

```
 \FOspaceleft=\pagegoal
 \advance\FOspaceleft by -\pagetotal
 \ifdim\FOspaceleft<\@tempdima \clearpage \fi}
```

Second pass: we use the actual height as target height. Why use a vbox here?

```
\def\jg@tablerow@secondpass#1{
  \NoTableCellHeight=\@tempdima
  \NoTableCheckHeighttrue
  \vbox{\hbox{#1}}}
```

The code looks like this. Note the insertion of the opposite of `\lineskip`. This assumes that TeX places a glue of value `\lineskip`, because the height of the box is greater than the baselineskip limit.

```
\def\NoTableRow#1{%
 \jg@default@cell@height
 \global\CellCount0 \jg@tablerow@firstpass{#1}%
 \ifdim\@tempdima>\NoTableCellHeight
   \global\CellCount0 \jg@tablerow@secondpass{#1}%
 \else
   \box0\relax
 \fi
 \vskip-\lineskip}
```

In the case of a table cell, we call some functions. A table body can consist of rows, or cells. In the case of cells, we have an attribute that says if the cell starts or ends a row.

```
\XMLelement{fo:table-cell}
  {
   \XMLattributeX{ends-row}{\FOendsrow}{false}
   \XMLattributeX{starts-row}{\FOstartsrow}{false}
  }
  {\xmlgrab}
  {\FOlabel
  \FOexpandattributes
  \NoTableCell{#1}}
```

This reduces the argument by the width of the padding, margin and border width on both sizes. Thus, we have in `#1`, a dimension register, the width of the region in which we can typeset the cell.

```
\def\jg@removemarg#1{
   \advance#1 by -\FOpaddingstart
   \advance#1 by -\FOpaddingend
   \ifx\FOborderstartstyle\att@solid\advance#1 by -\FOborderstartwidth\fi
   \ifx\FOborderendstyle\att@solid\advance#1 by -\FOborderendwidth\fi
   \advance#1 by -\FOmarginright
   \advance#1 by -\FOmarginleft}
```

If this cell is the first in a row, we emit a `\vskip`, whose value is minus `\lineskip` (this is the same action as when a column ends). In any case we set the cell count accordingly.

```
\def\jg@test@startrow{%
 \ifx\FOstartsrow\att@true
    \vskip-\lineskip
    \global\CellCount1
 \else
```

```
    \global\advance\CellCount by 1
 \fi}
```

If this cell is the last in a row, we emit a `\vskip` as above, and we set the counter to zero.

```
\def\jg@test@endrow{%
 \ifx\FOendsrow\att@true
    \vskip-\lineskip
    \global\CellCount0
 \fi}
```

Write `\CCWidth` instead of `\CurrentCellWidth` for simplicity. This is the width of the cell. If `\NCols` is zero, no specifications are given for the table. In this case, we use the natural width of the cell. Otherwise we use the stored value.

```
\def\jg@getCCwidth#1{
 \ifnum\NCols<1
  \CCWidth\z@
  \setbox0=\hbox{#1}%
  \CCWidth=\wd0
 \else
  \CCWidth=\csname fotable\the\AbsoluteTableCount:\the\CellCount\endcsname
 \fi}
```

If the cell spans more than one column, we assume that specifications are given for all columns. We compute the sum of these quantities.

```
\def\jg@getCCwidth@aux{%
 \ifnum\FOnumbercolumnsspanned>1
    \@tempcnta1
    \loop\ifnum\@tempcnta<\FOnumbercolumnsspanned
        \advance\@tempcnta by 1
        \global\advance\CellCount by 1
        \advance\CCWidth\csname fotable\the
           \AbsoluteTableCount:\the\CellCount\endcsname
    \repeat
 \fi}
```

This is now the code of a cell. The parent is a table row or a table. If we are in a row, all cells will be put in a hbox, and handled by the code shown above. Otherwise, we could be in vertical mode (because of the `\vskip` commands that separates rows), for this reason we insert a `\leavevmode`.

```
\def\NoTableCell#1{%
 \jg@test@startrow
 \jg@getCCwidth{#1}%
 \jg@removemarg{\CCWidth}
 \jg@getCCwidth@aux
 \ifx\XML@parent\FOTableRow
   \FOTableCellBlock#1\FOEndTableCellBlock
 \else
   \leavevmode\hbox{\FOTableCellBlock#1\FOEndTableCellBlock}%
 \fi
 \jg@test@endrow}
```

Some declarations.

```
\XMLNSA{fo}{column-width}{\FOcolumnwidth}{}
\XMLNSAX{fo}{number-columns-repeated}{\FOnumbercolumnsrepeated}{1}
```

```
\XMLNSAX{fo}{number-columns-spanned}{\FOnumbercolumnsspanned}{1}
\XMLNSAX{fo}{column-number}{\FOcolumnnumber}{}
```

4.9 Boxed blocks

We typeset a cell by opening in a lrbox (this is really a `\hbox`) in which we open a `\vbox`. We hack spacing. We removed a test to an undefined variable, that could center vertically the box.

```
\def\FOTableCellBlock{%
 \begin{lrbox}{\CellBox}%
 \vbox\bgroup
 \hsize\the\CurrentCellWidth
 \color@begingroup
 \FOSetFont{tablecellblock}%
 \jg@activew}
```

This is a bit longish, but easy to understand. We finish our `\vbox` and our lrbox. After that, we construct some boxes. Let's denote by PA, PB, PS, and PE the padding before, after (vertical), start, end (horizontal), by BA, BB, BS, and BE the border width (whenever the border is solid), and by ML, MR, MT and MB the margins (left, right, top, bottom). Let a be the sum of MT, PB and BB, and let b be the sum of PS, PE and the width of the cell box. The first box we construct is A, the cell box. The second box we construct is B, an hbox containing PS, A and PE. The third box is C, a vbox containing PB, fill, B, fill and PA. The fill is a `\vfil`, which is inserted in certain circumstances : if NoTableCheckHeight is true, we insert the filler if some attributes have the right value. In this case C is a `\vtop` to the height of the table computed earlier (this is the height of the row, we are in the second pass). The width of this box is the quantity b defined above. Now we construct a box D, an hbox with BS, bg, C, and BE, where bg may be empty if no background is desired. Otherwise it consists of a rule of width b, followed by a kern of width minus b (the height of the rule is the height of C). Then comes a box E, this is a vbox containing BB, D and BA. Note that BS, BE, BB and BA are borders: they are implemented via hrules or vrules. Then comes a box F, it is a vbox containing MT, E, and MB, and a box G, this is an hbox that contains ML, F, and MR. This box is shifted down by the quantity a. Finally, we put everything in an hbox.

```
\def\FOEndTableCellBlock{%
 \color@endgroup
 \egroup
 \end{lrbox}%
 \@tempdima\FOmargintop
 \advance\@tempdima\FOpaddingbefore
 \ifx\FOborderbeforestyle\att@solid\advance\@tempdima\FOborderbeforewidth\fi
\@tempdimb\wd\CellBox
\advance\@tempdimb by \FOpaddingstart
\advance\@tempdimb by \FOpaddingend
\hbox{%
    \lower\@tempdima
     \hbox{%
      \hskip\FOmarginleft
      \vbox{%
        \vskip\FOmargintop
        \vbox{%
        \ifx\FOborderbeforestyle\att@solid
           {\color{\FOborderbeforecolor}\hrule\@height\FOborderbeforewidth}%
        \fi
```

```
1034          \hbox{%
1035            \ifx\FOborderstartstyle\att@solid
1036              {\color{\FOborderstartcolor}\vrule\@width\FOborderstartwidth}%
1037            \fi
1038            \ifx\FObackgroundcolor\att@transparent
1039            \else
1040             {\color{\FObackgroundcolor}\vrule\@width\@tempdimb\kern-\@tempdimb}%
1041            \fi
1042             \ifNoTableCheckHeight
1043               \vtop to \NoTableCellHeight{%
1044                 \vskip\FOpaddingbefore
1045                 \ifx\FOdisplayalign\att@auto
1046                   \else\ifx\FOdisplayalign\att@after
1047                     \else\ifx\FOdisplayalign\att@before\vfil
1048                       \else\ifx\FOdisplayalign\att@centered\vfil\fi
1049                     \fi
1050                   \fi
1051                 \fi
1052                 \hbox{\kern\FOpaddingstart\box\CellBox\kern\FOpaddingend}%
1053                 \ifx\FOdisplayalign\att@auto\vfil
1054                   \else\ifx\FOdisplayalign\att@after\vfil
1055                     \else\ifx\FOdisplayalign\att@before
1056                       \else\ifx\FOdisplayalign\att@centered\vfil\fi
1057                     \fi
1058                   \fi
1059                 \fi
1060                 \vskip\FOpaddingafter
1061               }%
1062             \else
1063              \vbox{%
1064                \vskip\FOpaddingbefore
1065                  \hbox{\kern\FOpaddingstart\box\CellBox\kern\FOpaddingend}%
1066                \vskip\FOpaddingafter
1067                }%
1068             \fi
1069            \ifx\FOborderendstyle\att@solid
1070             {\color{\FOborderendcolor}\vrule\@width\FOborderendwidth}%
1071            \fi
1072          }%
1073          \ifx\FOborderafterstyle\att@solid
1074             {\color{\FOborderaftercolor}\hrule\@height\FOborderafterwidth}\fi
1075          }%
1076          \vskip\FOmarginbottom
1077       }%
1078       \hskip\FOmarginright
1079      }%
1080     }%
1081  }
```

This is done when we start a boxed object. This is like the start of a cell, but a bit simpler. Note the `\Quadding` and the `\strut`.[8]

[8] We hacked a bit this code.

```
\def\FOBoxedBlock#1{%
 \@tempdimb#1%
 \jg@removemarg{\@tempdimb}
 \begin{lrbox}{\BlockBox}%
 \vbox\bgroup
 \hsize\the\@tempdimb
 \FOSetFont{tableblock}%
 \color@begingroup
 \jg@activew
 \parindent\FOtextindent
 \Quadding
 \strut }
```

This is like the end of a cell box. It is a bit simpler. Dimensions a and b are renamed to b and c. We construct a box, as follows: A is the box that contains the material seen so far, B contains PS, A, and PE, C contains PB, B and PA, D contains BS, bg, C, and BE, E contains BD, D and BA, F contains MT, E and MB, finally G contains Ml, F and MR. The difference with a cell: no vertical skip, and Ml is margin-left plus text-indent[9].

```
\def\FOEndBoxedBlock{%
 \par
 \color@endgroup
 \egroup
 \end{lrbox}%
 \@tempdimb\FOmargintop
 \advance\@tempdimb\FOpaddingbefore
 \ifx\FOborderbeforestyle\att@solid\advance\@tempdimb\FOborderbeforewidth\fi
 \@tempdimc\wd\BlockBox
\advance\@tempdimc by \FOpaddingstart
\advance\@tempdimc by \FOpaddingend
\FOtempdim\FOmarginleft
\advance\FOtempdim by \FOtextindent
\hbox{%
    \lower\@tempdimb
     \hbox{%
      \kern\FOtempdim
      \vbox{%
        \vskip\FOmargintop
        \vbox{%
        \ifx\FOborderbeforestyle\att@solid
           {\color{\FOborderbeforecolor}\hrule\@height\FOborderbeforewidth}%
        \fi
        \hbox{%
          \ifx\FOborderstartstyle\att@solid
            {\color{\FOborderstartcolor}\vrule\@width\FOborderstartwidth}\fi
          \ifx\FObackgroundcolor\att@transparent
          \else
           {\color{\FObackgroundcolor}\vrule\@width\@tempdimc\kern-\@tempdimc}%
          \fi
            \vbox{%
              \vskip\FOpaddingbefore
               \hbox{\kern\FOpaddingstart\box\BlockBox\kern\FOpaddingend}%
```

[9] Why do we add the indentation?

```
                \vskip\FOpaddingafter
                }%
            \ifx\FOborderendstyle\att@solid
             {\color{\FOborderendcolor}\vrule\@width\FOborderendwidth}\fi
          }%
          \ifx\FOborderafterstyle\att@solid
             {\color{\FOborderaftercolor}\hrule\@height\FOborderafterwidth}\fi
          }%
          \vskip\FOmarginbottom
       }%
       \kern\FOmarginright
      }%
    }%
}
```

Attributes.

```
\XMLNSAX{fo}{display-align}{\FOdisplayalign}{\inherit}
\XMLstringX\att@top<>top</>
\gdef\FOdisplayalign{auto}
```

4.10 Lists

In XSL/Format four formatting objects construct lists: the main element is `<fo:list-block>`, the children are one or more `<fo:list-item>`, each containing a pair of `<fo:list-item-label>` and `<fo:list-item-body>`. There are two attributes, specific to lists:

```
\XMLNSAX{fo}{provisional-label-separation}
   {\FOprovisionallabelseparation}{\inherit}
\XMLNSAX{fo}{provisional-distance-between-starts}
   {\FOprovisionaldistancebetweenstarts}{\inherit}
```

This command is used twice: at the start of a list, and at the start of a block in a list. It defines some parameters that are used by the `\item` command.

```
\def\jg@use@listparams{
  \itemindent=\FOstartindent
  \leftmargin=\FOprovisionaldistancebetweenstarts
  \rightmargin=\FOmarginright
  \labelwidth=\FOprovisionaldistancebetweenstarts
  \advance\labelwidth by -\FOprovisionallabelseparation}
```

The translation of `<fo:list-block>` is essentially a call to the `list` environment. We have to redefine a lot of commands that deal with horizontal and vertical spacing. We globally change (increase by one) the FOinList counter. We call a command possibly defined via `\csname`. The `\expandafter` is useless.

```
\XMLelement{fo:list-block}
  {\SpaceAttributes}
  {
   \jg@hackindent
   \FOSetFont{normal}%
   \advance\FOinList by 1\relax
   \ifnum\FOinList>1\relax\leavevmode\fi
   \begin{list}{}{%
     \jg@use@listparams
```

```
      \advance\leftmargin by \FOmarginleft
      \expandafter\csname List\FOtextalign\endcsname
      \labelsep\FOprovisionallabelseparation
      \itemsep\z@ \parsep\z@ \topsep\z@  \parskip\z@
      \jg@usespacebefore{\topsep} }
    }
   {\end{list}
    \advance\FOinList by -1
    \par
   }
```

These are the definitions used in the code above. The default value (reset by `\list` in any case) corresponds to 'end'.

```
\def\Listjustified{  \gdef\makelabel##1{##1}}
\def\Liststart{      \gdef\makelabel##1{##1\hfil}}
\def\Listend{        \gdef\makelabel##1{\hfil##1}}
\def\Listcentered{   \gdef\makelabel##1{\hfil##1\hfil}}
\def\Listcenter{     \gdef\makelabel##1{\hfil##1\hfil}}
```

A list should contain `<fo:list-item>` elements. We insert some vertical space at the start of the item. In the case there is a label, we handle it after the `\vskip`.

```
\XMLelement{fo:list-item}
  {\SpaceAttributes}
  {
   \FOSetHyphenation
   \jg@usespacebefore{\vskip}
   \FOlabel}
  {}
```

An item is declared via `<fo:list-item-label>`. Question: why do we put the argument into `\ItemBox`? the box is never used! We shall see later what happens.

```
\newsavebox\ItemBox
\XMLelement{fo:list-item-label}
  {}
  {\xmlgrab}
  {\jg@hackindent
   \savebox{\ItemBox}{#1}\item[#1]\FOlabel}
```

The body of the item is declared via `<fo:list-item-body>`. We say that we are in the body of the list, and start a counter with zero.

```
\XMLelement{fo:list-item-body}
  {}
  {\ifx\FOstartindent\att@bodystart \let\FOstartindent\z@  \fi
   \FOListBodytrue\FOListBlocks0}
  {}
```

Default values for the attributes.

```
\gdef\FOstartindent{\z@}
\gdef\FOendindent{\z@}
\gdef\FOprovisionaldistancebetweenstarts{24.0pt}
\gdef\FOprovisionallabelseparation{6.0pt}
\newcount\FOListBlocks
```

4.11 Blocks

The <fo:block> is an important object.

We have seen cases where the switch \ifFOBlockGrab has to be set to true. Here is another one: when the border style is solid or when the background is not transparent.

```
\def\jg@hack@background{%
     \ifx\FObackgroundcolor\att@transparent
        \ifx\FOborderstyle\att@solid \FOBlockGrabtrue \fi
     \else
         \FOBlockGrabtrue
     \fi}
```

This piece of code sets \ifFOListInnerPar to true if we are inside a list and this is not the first block.

```
\def\jg@setlistinnerpar{%
        \ifx\XML@parent\FOListItemBody
          \global\advance\FOListBlocks by 1%
          \ifnum\FOListBlocks=1\relax\else\FOListInnerPartrue\fi
        \else
          \ifFOListBody \FOListInnerPartrue \fi
        \fi}
```

This is the action associated to a block.
We define two commands, \w@t and \@whattodonext, to be executed at the start of the element, and at the end. The first is followed by \jg@keep@together, the second is preceded by \jg@keep@together. This is a command that may evaluate a \samepage command. There are four cases to consider. Case 1: the parent is a FOListItemLabel. In this case, we execute \FOListBlock and \FOEndBlock. Case 2: FOInOutput is true. The actions are \FOOutputBlock and \FOEndOutputBlock. Case 3: In a table. Actions are \FOBoxedBlock and \FOEndBoxedBlock. Case 4: actions are \FONormalBlock and \FOendBlock. Remember, when we typeset a static block (page headers), we are in case 2, and the parent is empty; can this element contain lists? this would give strange page headings.

```
\XMLelement{fo:block}
  {\SpaceAttributes}
  {%
  \FOBlockGrabfalse
  \FOexpandattributes
  \FOSetHyphenation
  \ifx\XML@parent\FOListItemLabel
       \def\w@t{\FOListBlock}%
       \def\@whattodonext{\FOEndBlock}%
  \else
   \ifFOinOutput
     \jg@hack@background
     \def\w@t{\FOOutputBlock}%
     \def\@whattodonext{\FOEndOutputBlock}%
   \else
       \ifnum\FOinTable>\z@
         \def\w@t{\FOBoxedBlock{\CurrentCellWidth}}%
         \def\@whattodonext{\FOEndBoxedBlock}
       \else
        \jg@setlistinnerpar
```

```
          \jg@hack@background
          \def\@whattodonext{\FOEndBlock}
          \def\w@t{\FONormalBlock}%
        \fi
      \fi
    \fi
    \w@t
    \jg@keep@together
    }
    {\jg@keep@together
    \@whattodonext}
```

Declarations.

```
\XMLNSA{fo}{background-color}{\FObackgroundcolor}{transparent}
\XMLstringX\att@transparent<>transparent</>
\XMLstringX\att@solid<>solid</>
```

This is called in the case where the current block is the label of a list. We insert some vertical space (namely `\itemsep`), and compute the space needed for horizontal placement of the item.

```
\def\FOListBlock{%
   \FOSetFont{normal}%
   \get@external@font\xdef\FOlistlabelfont{\external@font}%
   \jg@usespacebefore{\itemsep}
   \jg@use@listparams
   \expandafter\csname List\FOtextalign\endcsname
}
```

This piece of code is executed at the end of a block (except output, or boxed). We do nothing if this is a label of a list (see later). We emit a `\par` if this block is somewhere else in a list and FOListInnerPar is true. We call `\FOEndBlockTwo` in the case where the block is neither in a list nor in a table.

```
\def\FOEndBlock{%
  \ifx\XML@parent\FOListItemLabel
  \else
    \ifnum\FOinList>0
      \ifFOListInnerPar\par\fi
    \else
       \ifnum\FOTableNesting>0
       \else \FOEndBlockTwo  \fi
    \fi
   \fi}
```

This is the end of a normal block. We first terminate the paragraph. After that, we call `\FOEndBoxedBlock` if the block is boxed, or else add some vertical space and some padding. After that, we decide if a new page, or a new double page should be started here. We compute in `\if@tempswa` a boolean value that says whether or not it is wise to start a new page here. If it is, we just insert some vertical space, otherwise, we insert a penalty, some vertical space, and call a command `\@afterheading` (a standard LaTeX command).

```
\def\FOEndBlockTwo{%
  \par
   \ifFOBlockGrab
       \FOEndBoxedBlock
   \else
        \ifdim\FOpaddingafter>\z@ \vskip\FOpaddingafter \fi
```

```
1272          \FOBorderBottom
1273     \fi
1274     \jg@handle@breakafter
1275     \jg@keepnext
1276     \if@tempswa\nobreak\fi
1277     \FOvspaceafter
1278     \if@tempswa\@afterheading\fi}
1279 \XMLstringX\att@oddpage<>odd-page</>
```

This considers the break-after attribute. the value can be 'auto', 'column', 'page', 'even-page' or 'odd-page'. . The case of even page is not considered. The normal page should use `\newpage`, rather than a simple penalty. The case of odd-page should use a blank page.

```
1280 \def\jg@handle@breakafter{%
1281    \ifx\FObreakafter\att@page \penalty -\@M
1282    \else \ifx\FObreakafter\att@oddpage \cleardoublepage \fi
1283    \fi}
```

In principle, a break before a block should be interpreted in the same way as a break after. Note that here, we have a blank page.

```
1284 \def\jg@handle@breakbefore{%
1285       \ifx\FObreakbefore\att@page
1286         \penalty -\@M
1287       \else
1288         \ifx\FObreakbefore\att@oddpage
1289         \penalty -\@M
1290         \ifodd\c@page\else\BlankPage\newpage\fi
1291         \fi
1292       \fi}
```

This is the start of an output block. It is not complicated. We call functions that specify how to align (justify) the text. If BlockGrab is true, we put the text in a box with a given size. Note: We added `\@inlabelfalse` because the `\color` command says `\leavevmode` if this is true. This gives an empty line if a pagebreak occurs because of an `\item`. We kill also `\FOid`. I'm not sure that this is useful, but the page head or foot should contain no anchor.

```
1293 \def\FOOutputBlock{%
1294  \FOSetFont{output}%
1295  \@inlabelfalse\let\FOId\@empty
1296  \ifFOBlockGrab \FOBoxedBlock{\textwidth}  \fi
1297  \QuaddingStart
1298  \Quadding}
```

The end code is easy. If we have opened a grab block, we must close it.

```
1299 \def\FOEndOutputBlock{%
1300  \QuaddingEnd
1301  \ifFOBlockGrab \FOEndBoxedBlock \fi
1302  \par}
```

This is used at the start of a border block. If the border is solid and has non-zero width, we insert the `\hrule`.

```
1303 \def\FOBorderTop{%
1304   \ifdim\FOborderbeforewidth>\z@
1305    \ifx\FOborderbeforestyle\att@solid
1306     {\color{\FObordercolor}\hrule height \FOborderbeforewidth}%
```

```
    \fi
 \fi}
```

This is used at the end of a border block. If the border is solid and has non-zero width, we insert the `\hrule`. Note: This may set FOBlockGrab to true. This is very strange (isn't it too late?).

```
\def\FOBorderBottom{%
  \ifx\FOborderafterstyle\att@solid
   \ifx\FOborderafterwidth\att@thin\def\FOborderafterwidth{0.4pt}\fi
   \ifx\FOborderafterwidth\att@medium\def\FOborderafterwidth{0.8pt}\fi
   \ifx\FOborderafterwidth\att@thick\def\FOborderafterwidth{1.2pt}\fi
  \else
   \def\FOborderafterwidth{\z@}%
  \fi
  \ifx\FOborderbeforestyle\att@solid
   \ifx\FOborderbeforewidth\att@thin\def\FOborderbeforewidth{0.4pt}\fi
   \ifx\FOborderbeforewidth\att@medium\def\FOborderbeforewidth{0.8pt}\fi
   \ifx\FOborderbeforewidth\att@thick\def\FOborderbeforewidth{1.2pt}\fi
   \FOBlockGrabtrue
  \else
   \def\FOborderbeforewidth{\z@}%
  \fi
   \ifdim\FOborderafterwidth>\z@
    \ifx\FOborderafterstyle\att@solid
      {\color{\FObordercolor}\hrule height \FOborderafterwidth}%
    \fi
   \fi
}
```

This will be used later. It redefines the behavior of space and end-of-line. The value of white-space can be 'normal', 'pre', or 'nowrap'. Its effect is to specify some other properties; first white-space-treatment, that can be 'ignore', 'preserve', 'ignore-if-before-linefeed', 'ignore-if-after-linefeed', 'ignore-if-surrounding-linefeed'; then white-space-collapse that can be 'true' or 'false'; then linefeed-treatment can be 'ignore', 'preserve', 'treat-as-space', 'treat-as-zero-width-space'; finally, wrap-option can be 'wrap' or 'no-wrap'. It is not clear to me if the code that follows has anything to do with the XSL/Format recommendations.

```
\def\jg@activew{%
   \ifx\FOwhitespace\att@pre\obeyspaces\obeylines\fi
   \ifx\FOwhitespacecollapse\att@false\obeyspaces\fi
   \ifx\FOwrapoption\att@nowrap\obeylines\fi}
```

We declare the attribute and the default value.

```
\XMLNSAX{fo}{wrap-option}{\FOwrapoption}{\inherit}
\XMLNSAX{fo}{white-space}{\FOwhitespace}{\inherit}
\XMLNSAX{fo}{white-space-collapse}{\FOwhitespacecollapse}{\inherit}
\gdef\FOwrapoption{wrap}
\gdef\FOwhitespace{normal}
\gdef\FOwhitespacecollapse{true}
\XMLstringX\att@nowrap<>no-wrap</>
```

This is what we do in the case of a normal block. The code was a bit simplified: We removed a reference to `\@x` (always `\relax`), the code executed in case of tables in tables (this cannot happen), and the code that saves `\FOid` if a page is started here.

If we are in a list, three actions are taken: a) we evaluate the label, b) emit a `\par` and some vertical space (provided that this is allowed), and c) look at translation of spaces and new lines. In the general case, more actions are taken. If there is an attribute that says that we must change page we do it. We emit a `\par`. If we must grab, we start a grab box. Otherwise we add some vertical space (note: a page break can occur here; in this case the label is on the wrong page; for this reason we moved the `\FOlabel` near the end). We define `\leftskip`, `\rightskip` and some other quantities. We execute action c) above.

```
\def\FONormalBlock{%
%  \ifnum\FOTableNesting>0
%  \else
    \ifnum\FOinList>0
      \FOlabel
      \ifFOListInnerPar\par\FOvspacebefore\fi
      \jg@activew
    \else
      \jg@handle@breakbefore
      \ifx\FObreakbefore\att@page
        \penalty -\@M
      \else
        \ifx\FObreakbefore\att@oddpage
        \penalty -\@M
        \ifodd\c@page\else\BlankPage\newpage\fi
        \fi
      \fi
      \par
%      \FOlabel JG
      \Quadding
      \ifFOBlockGrab
        \FOBoxedBlock{\linewidth}%
      \else
        \FOBorderTop
        \ifdim\FOpaddingbefore>\z@
         \vskip\FOpaddingbefore
        \fi
        \FOvspacebefore
        \parindent\FOtextindent
        \advance\leftskip by  \FOpaddingstart
        \advance\leftskip by  \FOmarginleft
        \advance\rightskip by \FOpaddingend
        \advance\rightskip by \FOmarginright
      \fi
      \jg@activew
    \fi
    \FOlabel %moved this [jg]
    \FOSetFont{normal}%
%  \fi
}
```

Attributes.

```
\XMLNSAX{fo}{break-before}{\FObreakbefore}{auto}
\XMLNSAX{fo}{break-after}{\FObreakafter}{auto}
```

We now redefine the `\item` command. This is a bit complicated. For this reason, we split the command into smaller pieces. The first idea is that the label is typeset in a box, and this box will be used later. The `\sbox` command produces essentially a hbox. The LaTeX source file says that the `\label` command should produce some glue, for instance, the code on lines 1174 to 1177 is OK, the code on line 1173 is not. The first line was added because it corrects a bug (this is strange, since essentially, `\savebox` is the same as `\sbox`).

```
\def\@itemD#1{%
  \savebox{\ItemBox}{#1}% Added Grimm
  \sbox\@tempboxa{\makelabel{#1}}}%
```

We can have more than one consecutive labels. For this reason, we put in the box `\@labels` all these labels. We consider three dimensions, say A, B, and C, that contain the space after the label, the nominal width of the label, and the total space allowed for the label and its surrounding space. We emit four items of glue, first, three items that correspond to $C - A - B$, then the label box then A. In the case where the width of the label is smaller than B, we use `\hbox to`, with a `\unhbox`: here it is important that the box contains glue that can stretch without producing underful boxes.

```
\def\@itemE{%
  \global\setbox\@labels\hbox{%
    \unhbox\@labels
    \hskip \itemindent
    \hskip -\labelwidth
    \hskip -\labelsep
    \ifdim \wd\@tempboxa >\labelwidth
      \box\@tempboxa
    \else
      \hbox to\labelwidth {\unhbox\@tempboxa}%
    \fi
    \hskip \labelsep}}%
```

In the case of an enumeration, there is a counter, whose name is in `\@listctr`, that has to be incremented. We know that we are in an enumeration, because `\if@nmbrlist` is true. In the case `\item` has an optional argument, `\if@noitemarg` is false, and in this case the user gives a number, so that the counter is not modified. Indexif"@noitemarg

```
\def\@itemC{%
  \if@noitemarg
    \@noitemargfalse
    \if@nmbrlist \refstepcounter\@listctr \fi
  \fi}
```

There is a switch, `\if@inlabel` "that is false except between the time an `\item` is encountered and the time that TeX actually enters horizontal mode", the quote is of Lamport. At the end of a list, or inside `\newpage`, if the switch is true, `\leavevmode` is executed, and the switch reset to false; this has as effect to flush pending labels. Otherwise, the switch is set to true in `\@itemA` shown below, and to false by `\@itemB`. There is another switch, `\if@noparitem`, it is set by `\list` to true if `\@inlabel` is true. This is the case when the use says, for instance `\item[foo] \begin{itemize}\item[bar]`. In this case, we have two items in a row. The code that follows handles this special case. The action is divided into three parts: first, we set the switch to false, second, we shift the labels left by the value of `\leftmargin` (note that this is how much the current list is indented with respect to its environment), finally, we adjust vertical space. We add two items of glue. In order to understand what happens, let A be the value of `\lastskip`. If the last item in the vertical list is glue, then A contains this value, otherwise it will be zero. Let B be the value of `\parskip` at the start of the current list (in our example, this is the `\parskip`

of the outer list, saved in `\@outerparskip`). Finally, let C be the current `\parskip`, a copy of `\parsep`. The items of glue we insert are $-A$ and $A+B-C$. When the label is finally printed, TEX will use the current `\parskip`, hence C. This gives a total of $A+B$: the old glue plus the old `\parskip`. The reason for inserting $-A$ is unclear to me.

```
1404 \def\@donoparitem{%
1405   \@noparitemfalse
1406   \global\setbox\@labels\hbox{\hskip -\leftmargin
1407                               \unhbox\@labels
1408                                \hskip \leftmargin}%
1409   \if@minipage\else
1410     \@tempskipa\lastskip
1411     \vskip -\lastskip
1412     \advance\@tempskipa\@outerparskip
1413     \advance\@tempskipa -\parskip
1414     \vskip\@tempskipa
1415   \fi}
```

The following piece of code is executed if `\if@newlist` is true. This quantity is set to true at the start of a list, and to false at the end (after signaling an error: empty list). It is set to false when the first label is printed (see `\@itemB` below). The effect of `\@nbitem` is to add some vertical space (the value is $B-C$, according to the description above). Otherwise, we use two `\addvspace` instead of a single one; instead of B, we use `\@topsep` (this is `\topsep` plus optional `\partopsep`). Comments added by fotex.sty. The LATEX source has a comment that says: Kludge if list follows `\section`.

```
1416 \def\@itemAA{%
1417 %       \if@nobreak \@nbitem \else  %%% REMOVED
1418          \addpenalty\@beginparpenalty
1419          \addvspace\@topsep
1420          \addvspace{-\parskip}%
1421 %       \fi
1422 }
```

This is now how `\item` manages vertical space. The code on line 1416 is strange: remember that `\indent` inserts current paragraph indentation, while `\par` terminates the current paragraph. On the next line, we have a double `\unskip`' the reason is that commands like `\addvspace` add a double skip in order to make it more difficult to remove it. However, this is generally not done in horizontal mode, strange...

```
1423 \def\@itemA{
1424   \if@noparitem
1425     \@donoparitem
1426   \else
1427     \if@inlabel \indent \par \fi
1428     \ifhmode \unskip\unskip \par \fi
1429     \if@newlist \@itemAA
1430     \else \addpenalty\@itempenalty \addvspace\itemsep \fi
1431     \global\@inlabeltrue
1432   \fi
1433 }
```

The `\fotex` file redefines `\item` like this. Note that `\leavevmode` makes sure the current mode is horizontal by starting a paragraph (remember that LATEX may redefine `\par`). What this code does is beyond me.

```
\let\olditem\item
\def\item{\if@inlabel\leavevmode\fi\olditem}
```

Now, the `\everypar` token list is redefined. The difference with the original LaTeX code is that some `\global` declarations have been added. In the case `\if@inlabel` is true, we set it to false, and output the label box. We kill `\itemindent`, in the case where the user says `\noindent` (in other cases, a new paragraph is created by an explicit or implicit `\indent`, that inserts a box of width `\parindent`).

```
\def\@itemB{%
 \global\everypar{%   %% Global added
    \@minipagefalse
    \global\@newlistfalse
    \if@inlabel
      \global\@inlabelfalse
      {\setbox\z@\lastbox
       \ifvoid\z@ \kern-\itemindent \fi}%
      \box\@labels
      \penalty\z@
    \fi
    \if@nobreak
      \global\@nobreakfalse   % Global added
      \clubpenalty \@M
    \else
      \clubpenalty \@clubpenalty
      \global\everypar{}%   % Global added
    \fi}}
```

This is now the code.

```
\def\@item[#1]{%
  \@itemA
  \@itemB
  \@itemC
  \@itemD{#1}%
  \@itemE
  \ignorespaces}
```

4.12 Other commands

4.12.1 Percentages

This command is a helper for percentage. If it is called with argument '`foo\relax%.\@{A}{B}`', and foo has no `%` character in it, then `#2` is a dot, the test is false, `B` is evaluated. If foo has the form `33.33%`, then the test is true, the number `33.33` is put in `\percentval` and `A` is evaluated.

```
{\catcode`\%=13
  \gdef\percenttest#1%#2#3\@{\ifx#2\relax
   \def\percentval{#1}\expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi}
```

This takes an argument, a dimension or a percentage. In the case of a dimension, it is put in `\@tempdima`. Otherwise it is converted. Conversion is strange: Assume the that number is `33.33`.

We put `33.33pt` in a skip register, divide this by 100, and use `\strip@pt`, a command that returns the value (without the unit, here 0.3333). This is then used as a multiplier for `\TableWidth`. The result is in `\SCALE`.

```
\gdef\TablePercentToDimen#1{\expandafter\percenttest#1\relax%.\@
  {\@tempdimb\percentval pt\relax\divide\@tempdimb by 100
   \edef\SCALE{\strip@pt\@tempdimb}\global\@tempdima
         =\SCALE\TableWidth}{\global\@tempdima#1}}
```

Same code, but we use `\hsize` instead of `\TableWidth`. The result is in `\SCALE`.

```
\gdef\PercentToDimen#1{\expandafter\percenttest#1\relax%.\@
  {\@tempdimb\percentval pt\relax\divide\@tempdimb by 100
   \edef\SCALE{\strip@pt\@tempdimb}\global\@tempdima
         =\SCALE\hsize}{\global\@tempdima#1}}
```

Same code, but we use `\Gin@nat@width`. We use the value of `\FOcontentwidth`. The result is in `\WSCALE`, is used for `\setkeys {Gin} {width=something}`.

```
\gdef\FOSetGWidth{\expandafter\percenttest\FOcontentwidth\relax%.\@
  {\@tempdima\percentval pt\relax\divide\@tempdima by 100
  \edef\WSCALE{\strip@pt\@tempdima}\setkeys{Gin}%
    {width=\WSCALE\Gin@nat@width}}{\setkeys{Gin}{width=\FOcontentwidth}}}
```

Same code, but we use `\Gin@nat@height` instead of `\TableWidth`. We use the value of `\FOcontentheight`. The result is in `\HSCALE`, used for `\setkeys {Gin} {height=something}.`

```
\gdef\FOSetGHeight{\expandafter\percenttest\FOcontentheight\relax%.\@
  {\@tempdima\percentval pt\relax\divide\@tempdima by 100
  \edef\HSCALE{\strip@pt\@tempdima}\setkeys{Gin}%
     {height=\HSCALE\Gin@nat@height}}{\setkeys{Gin}{height=\FOcontentheight}}}
```

Same idea. However, we divide `\f@size` by 100. The result is in `\FOfontsizefinal`.

```
\gdef\PlayWithFSize#1{\@default\f@size pt
  \expandafter\percenttest#1\relax%.\@
      {\dimen@0.01\@default
     \multiply\dimen@\percentval\relax}{\dimen@#1}%
     \edef\FOfontsizefinal{\the\dimen@}}
```

Same code, but we use `\baselineskip` instead of `\TableWidth`. Result in `\dimen@`.

```
\gdef\PlayWithShift{\expandafter\percenttest\FOverticalalign\relax%.\@
      {\dimen@0.01\baselineskip\multiply\dimen@\percentval\relax}%
   {\dimen@\FOverticalalign}}
} % Now % is a comment char again
```

4.12.2 Fonts

We show here some commands that deal with fonts. This is bit longish... This defines some values for the sizes.

```
\expandafter\def\csname size-xx-small\endcsname{7pt}
\expandafter\def\csname size-x-small\endcsname{8pt}
\expandafter\def\csname size-small\endcsname{9pt}
\expandafter\def\csname size-medium\endcsname{10pt}
\expandafter\def\csname size-large\endcsname{14.4pt}
\expandafter\def\csname size-x-large\endcsname{18pt}
\expandafter\def\csname size-xx-large\endcsname{20pt}
```

This puts in `\FOfontsizefinal` a size (could be 10pt, 100% or medium).

```
\def\computeFOfontsize{%
  \expandafter\ifx\csname size-\FOfontsize\endcsname\relax
    \PlayWithFSize\FOfontsize
  \else
    \edef\FOfontsizefinal{\csname size-\FOfontsize\endcsname}%
  \fi}
```

The mlnames.tex file defines `\Family@foo`, for different values of `foo`, for instance monospace, sansserif, serif, or Arial, Helvetica, or Avant-Garde, New-Century-Schoolbook, or cmr, cmss, etc.

```
\def\FOSetFont#1{%
 \FOSetHyphenation
 \edef\LaTeXshape{\csname Width@\FOfontstretch\endcsname
      \csname Weight@\FOfontweight\endcsname}%
 \ifx\LaTeXshape\@empty\def\LaTeXshape{m}\fi
 \edef\fFamName{\FOfontfamily}%
 \edef\f@series{\LaTeXshape}%
 \edef\f@shape{\csname Posture@\FOfontstyle\endcsname}%
 \ifx\FOfontvariant\att@smallcaps \def\f@shape{sc}\fi
 \let\f@family\relax
 \@for\FOfoo:=\FOfontfamily\do{%
    \ifx\f@family\relax
    \expandafter\let\expandafter\f@family
         \csname Family@\FOfoo\endcsname
    \fi}%
    \ifx\f@family\relax
      \def\f@family{\csname Family@\Defaultx@fontfamily\endcsname}%
    \fi
 \FOSetFontSize
 \selectfont
 \ifx\FOcolor\@empty \else \color{\FOcolor}\fi
}
```

This command computes some parameters for font changing. If no line-height is given, we use the font size plus twenty percent.

```
\def\FOSetFontSize{%
 \computeFOfontsize
 \ifx\FOlineheight\att@normal
   \@tempdima\FOfontsizefinal
   \multiply\@tempdima by 12
   \divide\@tempdima by 10
   \set@fontsize\baselinestretch{\FOfontsizefinal}{\@tempdima}%
 \else
   \set@fontsize\baselinestretch{\FOfontsizefinal}{\FOlineheight}%
 \fi}
```

Declarations and attributes.

```
\XMLNSAX{fo}{line-height}{\FOlineheight}{\inherit}
\XMLNSAX{fo}{font-weight}{\FOfontweight}{\inherit}
\XMLNSAX{fo}{font}{\FOfont}{\inherit}
\XMLNSAX{fo}{font-family}{\FOfontfamily}{\inherit}
\XMLNSA{fo}{font-size}{\FOfontsize}{\inherit}
\XMLNSAX{fo}{font-size-adjust}{\FOfontsizeadjust}{\inherit}
\XMLNSAX{fo}{font-stretch}{\FOfontstretch}{\inherit}
```

```
\XMLNSAX{fo}{font-style}{\FOfontstyle}{\inherit}
\XMLNSAX{fo}{font-variant}{\FOfontvariant}{\inherit}
\gdef\FOlineheight{normal}
\gdef\FOfontweight{normal}
\gdef\Defaultx@fontfamily{Times-Roman}
\gdef\FOfont{}
\gdef\FOfontfamily{Times-Roman}
\gdef\FOfontsizeadjust{none}
\gdef\FOfontsize{medium}
\gdef\FOfontstretch{normal}
\gdef\FOfontstyle{normal}
\gdef\FOfontvariant{normal}
\XMLstringX\att@smallcaps<>small-caps</>
```

All definitions in the remaining of this subsection come from mlnames.sty. We start with the Family.

```
\def\Family@monospace{pcr}     \def\Family@sansserif{phv}
\expandafter\def\csname Family@sans-serif\endcsname{phv}
\def\Family@serif{ptm}         \def\Family@cursive{uzc}
\def\Family@fantasy{uzc}       \def\Family@unknown{<unknown>}
\def\Family@Arial{phv}         \def\Family@Helvetica{phv}
\def\Family@Palatino{ppl}      \def\Family@Bookman{pbk}
\def\Family@BaskervilleMT{mbv} \def\Family@Courier{pcr}
\def\Family@Symbol{psy}        \def\Family@Wingdings{pzd}
\def\Family@WingDings{pzd}     \def\Family@LucidaSans{hls}
\def\Family@LucidaBright{hlh}  \def\Family@LucidaTypewriter{hlst}
\def\Family@Savoy{usb}         \def\Family@Luxi{ul9}
\def\Family@ACaslon{pca}       \def\Family@Caslon{uca}
\def\Family@Formata{pfa}       \def\Family@FranklinGothic{pfg}
\def\Family@OCRAbyBT{boa}      \def\Family@AGaramond{pad}
\expandafter\def\csname Family@Avant-Garde\endcsname{pag}
\expandafter\def\csname Family@Courier New\endcsname{pcr}
\expandafter\def\csname Family@New-Century-Schoolbook\endcsname{pnc}
\expandafter\def\csname Family@Times-Roman\endcsname{ptm}
\expandafter\def\csname Family@Trade-Gothic\endcsname{ptg}
\expandafter\def\csname Family@Times-New-Roman\endcsname{ptm}
\expandafter\def\csname Family@Times New Roman\endcsname{ptm}
\expandafter\def\csname Family@Times Roman\endcsname{ptm}
\expandafter\def\csname Family@Times-NR-MT\endcsname{mnt}
\expandafter\def\csname Family@Courier-New\endcsname{pcr}
\expandafter\def\csname Family@Zapf-Dingbats\endcsname{pzd}
\expandafter\def\csname Family@Gill-Sans\endcsname{pgs}
\expandafter\def\csname Family@iso-serif\endcsname{ptm}
\expandafter\def\csname Family@sans-serif\endcsname{phv}
\expandafter\def\csname Family@iso-sanserif\endcsname{phv}
\expandafter\def\csname Family@iso-monospace\endcsname{pcr}
\expandafter\def\csname Family@LetterGothic12PitchBT\endcsname{blg}
\expandafter\def\csname Family@NewsGothic\endcsname{bng}
\expandafter\def\csname Family@NewsGothicBT\endcsname{bng}
\expandafter\def\csname Family@Humanist521\endcsname{bgs}
\expandafter\def\csname Family@Humanist521BT\endcsname{bgs}
\expandafter\def\csname Family@Monospace821\endcsname{bhvt}
\expandafter\def\csname Family@Monospace821BT\endcsname{bhvt}
\expandafter\def\csname Family@OCRB10PitchBT\endcsname{bob}
\expandafter\def\csname Family@OCR-A\endcsname{boa}
\expandafter\def\csname Family@OCR-B-10PitchBT\endcsname{bob}
```

```
\expandafter\def\csname Family@Computer-Modern-Typewriter\endcsname{aett}
\expandafter\def\csname Family@Computer-Modern-Sans\endcsname{aess}
\expandafter\def\csname Family@Computer-Modern\endcsname{aer}
\expandafter\def\csname Family@Computer-Modern-Caps-And-Small-Caps\endcsname
{cmcsc}
\def\Family@cmr{cmr}              \def\Family@cmss{cmss}
\def\Family@cmtt{cmtt}            \def\Family@cmcsc{cmcsc}
\def\Family@ectt{ectt}            \def\Family@Utopia{put}
\def\Family@ZapfChancery{pzc}     \def\Family@Fibonacci{cmfib}
\def\Family@Funny{cmfr}           \def\Family@Dunhill{cmdh}
\def\Family@Concrete{ccr}         \def\Family@Charter{bch}
\def\Family@Fontpxr{pxr}          \def\Family@Fontaer{aer}
\def\Family@Fontaess{aess}        \def\Family@Fontaett{aett}
\def\Family@Fontlcmss{lcmss}      \def\Family@Fontlcmtt{lcmtt}
\def\Family@Fontcmvtt{cmvtt}      \def\Family@Fontcmbr{cmbr}
\def\Family@Fontcmtl{cmtl}        \def\Family@Fontpxss{pxss}
\def\Family@Fonttxss{txss}        \def\Family@Fonttxr{txr}
```

We define some postures.

```
\def\Posture@upright{n}
\def\Posture@normal{n}
\def\Posture@math{it}
\def\Posture@oblique{sl}
\def\Posture@backslantedoblique{ui}
\def\Posture@italic{it}
\def\Posture@backslanteditalic{ui}
```

We define some weights.

```
\def\Weight@ultralight{ul}
\def\Weight@extralight{el}
\def\Weight@light{l}
\def\Weight@semilight{sl}
\def\Weight@medium{}
\def\Weight@normal{}
\def\Weight@semibold{sb}
\def\Weight@bold{bx}
\def\Weight@extrabold{eb}
\def\Weight@ultrabold{ub}
\def\Weight@false{}
```

We define some widths.

```
\expandafter\def\csname Width@ultra-condensed\endcsname{uc}
\expandafter\def\csname Width@extra-condensed\endcsname{ec}
\expandafter\def\csname Width@condensed\endcsname{c}
\expandafter\def\csname Width@semi-condensed\endcsname{sc}
\expandafter\def\csname Width@normal\endcsname{}
\expandafter\def\csname Width@semi-expanded\endcsname{sx}
\expandafter\def\csname Width@expanded\endcsname{x}
\expandafter\def\csname Width@extra-expanded\endcsname{ex}
\expandafter\def\csname Width@ultra-expanded\endcsname{ux}
\def\Width@ultracondensed{uc}
\def\Width@extracondensed{ec}
\def\Width@condensed{c}
\def\Width@semicondensed{sc}
\def\Width@medium{}
\def\Width@semiexpanded{sx}
\def\Width@expanded{x}
```

```
\def\Width@extraexpanded{ex}
\def\Width@ultraexpanded{ux}
```

4.12.3 Links

The XSL/Format standard defines three attributes: target-presentation-context, target-processing-context, and target-stylesheet, but they are ignored. The code make use of a variable that is not defined. We removed the test. I don't understand these two `\let\xx\empty`, hence commented them out.

```
\XMLelement{fo:basic-link}
  { }
  {\xmlgrab}
  {
  \ifx\FOverticalalign\att@auto
     \let\FOverticalalign\FObaselineshift
  \fi
   \protectCS{\FOinternaldestination}%
   \protectCS{\FOexternaldestination}%
   \ifx\FOexternaldestination\@empty
    \hyperlink{\FOinternaldestination}{\FO@inlinesequence{#1}}%
    %\let\FOinternaldestination\@empty  %% JG
   \else
      \href{\FOexternaldestination}{\FO@inlinesequence{#1}}
    %\let\FOexternaldestination\@empty  %% JG
    \fi
  }
```

These are the attributes used above.

```
\XMLNSAX{fo}{external-destination}{\FOexternaldestination}{}
\XMLNSAX{fo}{internal-destination}{\FOinternaldestination}{}
```

4.12.4 Footnotes

Handling of `<fo:footnote>`*mark text*`</fo:footnote>`. We use two functions.

```
\XMLelement{fo:footnote}
  {}
  {\xmlgrab}
  {\xmltextwochildren\FOplainfootmark\FOplainfoottext#1}
```

This is transparent.

```
\XMLelement{fo:footnote-body}
  {}  {}   {}
```

There are two version, one is unused. This is the version actually used.

```
\def\FOplainfootmark#1{#1}

\long\def\FOplainfoottext#1{\insert\footins{%
    \reset@font\footnotesize
    \interlinepenalty\interfootnotelinepenalty
    \splittopskip\footnotesep
    \splitmaxdepth \dp\strutbox \floatingpenalty \@MM
    \hsize\columnwidth \@parboxrestore
```

```
    \color@begingroup
    #1\unskip\ifhmode\nobreak\fi\vskip\lineskip
    \color@endgroup}}
```

4.12.5 Inline material

Handling of `<fo:inline att>body</fo:inline>`. We call one of two alternatives, depending on whether the border is solid or not. As usual, we evaluate 'thin', 'thick' or 'medium'.

```
\XMLelement{fo:inline}
  {}
  {\xmlgrab}
  {
  \ifx\FOverticalalign\att@auto \let\FOverticalalign\FObaselineshift \fi
  \FOlabel
  \ifx\FOborderstyle\att@solid
   \ifx\FOborderwidth\att@thin\def\FOborderwidth{0.4pt}\fi
   \ifx\FOborderwidth\att@medium\def\FOborderwidth{0.8pt}\fi
   \ifx\FOborderwidth\att@thick\def\FOborderwidth{1.2pt}\fi
   \FOboxedsequence{#1}%
  \else
      \FO@inlinesequence{#1}%
  \fi}
```

Normal (non-boxed) inline sequence. We take into account two parameters: the decoration, and the vertical alignment. The alignment can be 'baseline', this is the normal behavior; it can be 'super' or 'sub', case where we raise or lower using special command; otherwise alignment is a dimension (or a percentage), and we use `\raisebox` to raise the box. We use `\csname` for the decoration. Question: do we really need the `\FOlabel` command after the `\csname`? Maybe we could put it *after* the last `\fi`.

```
\def\FO@inlinesequence#1{%
 \FOSetFont{normal}%
\ifx\FOverticalalign\att@baseline
  \csname DECO@\FOtextdecoration\endcsname{\FOlabel#1}%
\else
 \ifx\FOverticalalign\att@super
   \textsuperscript{\csname DECO@\FOtextdecoration\endcsname{\FOlabel#1}}%
 \else
    \ifx\FOverticalalign\att@sub
      \textsubscript{\csname DECO@\FOtextdecoration\endcsname{\FOlabel#1}}%
    \else
      \PlayWithShift
      \raisebox{\dimen@}{\csname DECO@\FOtextdecoration\endcsname{\FOlabel#1}}%
    \fi
 \fi
\fi
}
```

The original code uses two booleans `\ifFOSuper` and `\ifFOSub`. They are never defined, hence we removed the code that uses them. However, the code should be the same as before, with a frame. So, is this a bug?

```
\def\FOboxedsequence#1{%
  \FOSetFont{normal}%
```

```
  \ifx\FOborderwidth\@empty\else\fboxrule\FOborderwidth\fi
  \ifx\FOverticalalign\att@baseline
     \fbox{\csname DECO@\FOtextdecoration\endcsname{\FOlabel#1}}%
  \else
       \PlayWithShift \fbox{\raisebox{\dimen@}{\FOlabel#1}}%
  \fi}
```

This declares the attribute.

```
\XMLNSAX{fo}{text-decoration}{\FOtextdecoration}{none}
\XMLNSAX{fo}{baseline-shift}{\FObaselineshift}{baseline}
\XMLstringX\att@sub<>sub</>
\XMLstringX\att@super<>super</>
\XMLstringX\att@baseline<>baseline</>
```

The following decorations are known. Notice the first: if no decoration is given, the argument is typeset, without the braces.

```
\def\DECO@{\@firstofone}
\def\DECO@blink{\uwave}
\def\DECO@underline{\uline}
\expandafter\def\csname DECO@line-through\endcsname{\sout}
```

4.12.6 Floats and images

Support for .gif format. Is this needed?

```
\g@addto@macro\Gin@extensions{,.gif}
\@namedef{Gin@rule@.gif}#1{{png}{.png}{`giftopng #1}}
```

The fotex file defines Gin/scale in the same way as graphicx.sty. We do not repeat the code here.

```
\define@key{Gin}{scale}{ ... }
```

Comment by S. Rahtz: "Taken from Heiko Oberdiek's epstopdf.sty but the redefinitions need to be global". This allows on the fly conversions from one image type to another. For instance, this may call `giftopng`.

```
\global\let\orgGin@setfile\Gin@setfile
\global\def\Gin@setfile#1#2#3{%
  \if`\@car #3\relax\@nil
    \let\Gin@base\filename@base
    \immediate\write18{\@cdr #3\@empty\@nil}%
    \orgGin@setfile{#1}{#2}{\filename@base #2}%
  \else
    \orgGin@setfile{#1}{#2}{#3}%
  \fi}
```

This is trivial.

```
\XMLelement{fo:float}
  {\XMLattributeX{float}{\FOfloat}{float}}
  {\ifx\FOfloat\att@none \begin{figure}[!htp] \else  \begin{figure} \fi
   \FOlabel}
  {\end{figure}}
```

The following is a bit more complicated. One problem is that we have to merge content-height and height, the same for the width. Question: is this the desired result? In order to make the code easier to understand, we have added three auxiliary functions.

```
\def\jg@marge@aux{%
    \ifx\FOwidth\att@auto
      \ifx\FOheight\att@auto        \relax
      \else
        \def\FOcontentheight{\FOheight}%
        \def\FOcontentwidth{auto}%
       \fi
    \else
      \def\FOcontentwidth{\FOwidth}%
      \ifx\FOheight\att@auto
        \def\FOcontentheight{auto}%
      \else
        \def\FOcontentheight{\FOheight}%
      \fi
    \fi
}
\def\jg@mergeW{%
    \ifx\FOcontentwidth\att@scaletofit
      \setkeys{Gin}{width=\linewidth}%
    \else
      \ifx\FOcontentwidth\att@auto\else\FOSetGWidth\fi
    \fi
}
\def\jg@mergeH{%
    \ifx\FOcontentheight\att@scaletofit
       \setkeys{Gin}{height=\textheight}%
    \else
       \ifx\FOcontentheight\att@auto\else\FOSetGHeight\fi
    \fi
}
```

This piece of code considers text-align but we could also take into account display-align, that can be 'before', 'after' or 'center' (as well as 'auto'). Note that text-align can be 'start' or 'end' as well.

If scaling is 'uniform', the aspect ration should be preserved; is the default value of the Gin variable true?

```
\XMLelement{fo:external-graphic}
   {
      \XMLattribute{content-height}{\FOcontentheight}{auto}
      \XMLattribute{content-width}{\FOcontentwidth}{auto}
   }
   {
    \jg@filetest
    \jg@merge@aux
    \jg@mergeW
    \jg@mergeH
    \def\aligntype{center}
    \ifthenelse{\equal{\FOtextalign}{right}}{\def\aligntype{flushright}}{}
    \ifthenelse{\equal{\FOtextalign}{left}}{\def\aligntype{flushleft}}{}
    \def\Picscaled{\begin{\aligntype}%
       \includegraphics{\FOsrcname}\end{\aligntype}}
    \ifx\FOscaling\att@uniform\else\setkeys{Gin}{keepaspectratio=false}\fi
```

```
1797      \ifx\FOborderstyle\att@solid\fbox{\Picscaled}\else\Picscaled\fi
1798      }
1799      {}
```

Attributes used here.

```
1800  \XMLNSAX{fo}{scaling}{\FOscaling}{uniform}
1801  \XMLNSAX{fo}{height}{\FOheight}{auto}
1802  \XMLstringX\att@scaletofit<>scale-to-fit</>
1803  \XMLstringX\att@uniform<>uniform</>
```

This makes the string canonical. This is used in some places; it has an alternate name elsewhere.

```
1804  \def\protectCS#1{%
1805   \begingroup
1806         \utfeight@protect@chars
1807         \xdef\FOtempCS{#1}%
1808   \endgroup
1809   \let#1\FOtempCS}%
```

This function does something if the argument starts with 'url'. Otherwise, it removes a prefix of the form '`file://`' or '`file:`'. The result is in `\FOsrcname`.

```
1810  \def\FOfiletest#1#2#3#4#5#6#7#8\@{%
1811    \def\@tempa{#1#2#3#4#5#6#7}%
1812    \def\@tempb{#1#2#3#4#5}%
1813    \def\@tempc{#1#2#3#4}%
1814    \ifx\@tempa\file@prefix
1815      \xdef\FOsrcname{#8}%
1816    \else
1817     \ifx\@tempb\file@shortprefix
1818        \xdef\FOsrcname{#6#7#8}%
1819     \else
1820      \ifx\@tempc\file@urlprefix
1821      %%   \expandafter\FOurlfiletest#5#6#7#8%
1822      %% \@empty\@empty\@empty\@empty\@empty\@empty\@empty\@empty\@empty\@empty
1823      \jg@filetestaux#5#6#7#8)\@
1824      \else
1825        \xdef\FOsrcname{#1#2#3#4#5#6#7#8}%
1826      \fi  \fi  \fi}
```

These lines were added instead of lines 1821 and 1822. Guess why.

```
1827  \def\jg@filetestaux#1)#2\@{%
1828    \FOurlfiletest#1\@empty\@empty\@empty\@empty\@empty\@empty\@empty\@empty)}
```

Version with all the arguments.

```
1829  \def\jg@filetest{%
1830     {\utfeight@protect@chars
1831      \expandafter\FOfiletest\FOsrc\@empty
1832         \@empty\@empty\@empty\@empty\@empty\@empty\@}}
```

This function puts foo in `\FOsrcname` if the argument is `url(file://foo)` or `url(file:foo)` or `url(foo)` (the prefix '`url(`' has already been removed).

```
1833  \def\FOurlfiletest#1#2#3#4#5#6#7#8){%
1834    \def\@tempa{#1#2#3#4#5#6#7}%
1835    \def\@tempb{#1#2#3#4#5}%
1836    \ifx\file@prefix\@tempa
1837      \xdef\FOsrcname{#8}%
```

```
  \else
   \ifx\@tempb\file@shortprefix
      \xdef\FOsrcname{#6#7#8}%
   \else
      \xdef\FOsrcname{#1#2#3#4#5#6#7#8}%
   \fi \fi}
```

These are the strings needed by the function above.

```
\XMLstring\file@urlprefix<>url(</>
\XMLstring\file@prefix<>file://</>
\XMLstring\file@shortprefix<>file:</>
```

4.12.7 Markers

We consider here a list of the form X1Y1X2Y2...XnYn. The list ends with Xn being `\relax`. This command puts in the token list `\toks@` all pairs `{Xi}{Yi}` for which `Xi` is not equal to the value of the attribute marker-class-name. The name of the command is misleading; after this command has been evaluated, we are ready to add a marker.

```
\gdef\FOaddmarker#1#2{%
 \ifx\relax#1
 \else
   \def\FOtemp{#1}%
   \ifx\FOtemp\FOmarkerclassname
   \else \toks@\expandafter{\the\toks@{#1}{#2}}  \fi
   \expandafter\FOaddmarker
 \fi}
```

Handling of `<fo:marker marker-class-name=X>mark</fo:marker>`. We consider the content of `\FOmarks`, apply the command defined above, then add a new pair at the end; the pair is defined by *X* and *mark*. We copy the token list into the `\FOmarks` command (using full expansion, assignment is global), and then put it in a TeX mark.

```
\XMLelement{fo:marker}
 {}
 {\xmlgrab}
 {\toks@{}%
  \expandafter\FOaddmarker\FOmarks\relax{}%
  \toks@\expandafter{\the\expandafter\toks@\expandafter{\FOmarkerclassname}{#1}}%
  \xdef\FOmarks{\the\toks@}%
  \mark{\FOmarks}}
```

Here we assume that the list is terminated by a double `\relax`. The command finds and typesets the value associated to `\FOthisretrieveclassname`. In fact, we assume that `Xi` is in `#1`, `Yi` in `#2`. If the first argument is `\relax`, this is the end of the list, and we are done. If it is not the desired marker, we continue. Otherwise, we call a command that reads everything up to the double `\relax` at high speed. This will first read `{#2}`, this is `Yi` with a pair of additional braces that will disappear as argument `#1` of the next routine. It will then read `\fi`, two tokens, a second `\fi` and the list. The two `\fi` tokens are read again.

```
\gdef\FOgetmarker#1#2{%
  \ifx\relax#1
  \else
   \def\FOtemp{#1}%
    \ifx\FOtemp\FOthisretrieveclassname
      \FOmarkergobble{#2}%
```

```
      \fi
    \expandafter\FOgetmarker
  \fi}
\gdef\FOmarkergobble#1#2\relax\relax{\fi\fi#1}
```

Handling of `<fo:retrieve-marker retrieve-class-name=X retrieve-position=Y/>`. We fetch one of the marks (`\topmark` or `\botmark`), and extract what is needed.

```
\XMLelement{fo:retrieve-marker}
 {}
 {\xmlgrab}
 {\begingroup
    \utfeight@protect@chars
    \ifx\FOretrieveposition\FirstOnPage
       \xdef\FOthismark{\topmark}%
    \else \ifx\FOretrieveposition\LastOnPage
       \xdef\FOthismark{\botmark}%
    \else
       \xdef\FOthismark{\topmark}%
    \fi \fi
    \xdef\FOthisretrieveclassname{\FOretrieveclassname}
    \endgroup
    \expandafter\FOgetmarker\FOthismark\relax\relax}
```

Constants and attributes used in the code above.

```
\XMLNSAX{fo}{retrieve-class-name}{\FOretrieveclassname}{}
\XMLNSAX{fo}{retrieve-position}
     {\FOretrieveposition}{first-starting-within-page}
\XMLNSAX{fo}{marker-class-name}{\FOmarkerclassname}{}
\XMLstring\FirstOnPage<>first-starting-within-page</>
\XMLstring\LastOnPage<>last-starting-within-page</>
\gdef\FOmarks{}
```

4.12.8 Page numbers

Handling of `<fo:page-number-citation ref-id=A/>`. We just call `\pageref`.

```
\XMLelement{fo:page-number-citation}
   {\XMLattributeX{ref-id}{\FOrefid}{}}
   {}
   {\fopagecitation}
\def\fopagecitation{\pageref{\FOrefid}}
```

Handling of `<fotex:sort>...</fotex:sort>`. The content is a sequence of page-number-citation elements; we want to typeset the numbers, in increasing order, removing duplicates, and compacting them. The idea is the following: We redefine `\fopagecitation`, the command that handles the citations, it will construct a sorted list. When the end tag is seen, we shall compact and print the list.

```
\XMLelement{fotex:sort}
   {}
   {\let\fopagecitation\fosortpagecitation
    \global\sorttoks{}}
   {\global\sortcount-2\let\@elt\focompress@elt
    \let\fosep\@empty
    \let\foheld\relax
```

```
1907      \the\sorttoks
1908      \foheld}
```

We are faced here with the following problem: insert a number in an ordered list (increasing order). Assume that the list contains X_1, X_2, X_3, etc., and the number to insert is Y. The idea is to consider each X_i, renaming it locally X. There are three cases to consider. If $X < Y$, it is too early, if $X = Y$ we do nothing, if $X > Y$ we insert Y here. Note that, if $X_i > Y$, then $X_{i+1} > Y$. As a consequence, we shall replace Y by ∞ after insertion. At the end of the loop, we can consult the value of Y: if it is not ∞, we have to insert it at the end.

In practice, we proceed as follows. First, each X_i is of the form `\@elt{3}` or `\@elt{12}`. These quantities are in the token list `\sorttoks`. We shall see in a minute how to evaluate this list in such a way that, at the start of the evaluation, the list is empty. Moreover we change the meaning of `\@elt` to `\fosort@elt`. The quantity Y is in the counter `\sortcount`. In the case $X < Y$, we have to re-insert `\@elt{X}` in the list (line 1918). In the case $X > Y$, we insert `\@elt{Y}` and `\@elt{X}` in the list. The non trivial point is to evaluate Y (there is no problem for X, since this is in `#1`), this requires four `\expandafter` commands on lines 1913 and 1914. Remember: since `\sorttoks` is a reference to a token list, if you say `\global\sorttoks\foo`, then `\foo` is expanded, but the tokens in the list are not. Here we have `\expandafter` instead of `\foo`, so that the `\the` after the brace is expanded. Since `\the` fully expands what follows, its effect is first to expand the `\expandafter` (and as a consequence the next `\the`) and second, to replace `\sorttoks` by its value (this is a bit unusual).

```
1909 \newcount\sortcount
1910 \newtoks\sorttoks
1911 \def\fosort@elt#1{%
1912  \ifnum#1>\sortcount
1913    \global\sorttoks\expandafter{\the\expandafter\sorttoks\expandafter\@elt
1914        \expandafter{\the\sortcount}\@elt{#1}}%
1915   \global\sortcount\maxdimen
1916   \else
1917    \ifnum#1<\sortcount
1918    \global\sorttoks\expandafter{\the\sorttoks\@elt{#1}}%
1919    \fi
1920  \fi}
```

The `\fosortpagecitation` is used to add a page reference into a sorted list. Assume that `\label{foo}` has been executed somewhere. This defines `\r@foo`, to be a command that can be used by `\ref` or `\pageref`. In what follows, we want the number used by `\pageref`. For the case where `\pageref` is used before `\label`, LaTeX provides a mechanism in which information is printed in the auxiliary file, and read at the start of the job (and also at the end; sometimes you will see 'labels may have changed'). If you compile a file for the first time, the `\r@foo` command is undefined, and `\pageref` has to consider this case; this is the reason why we cannot use this command directly. We shall assume here that the hyperref package has been loaded, so that, when defined, `\r@foo` contains a list of five items, the second one being the page number. On line 1923 we call the command that returns this second argument. On the line that follows, we insert five `\relax` tokens, in order to make sure that there are at least five tokens. On the preceding line, there are three `\expandafter`: if we had only one, the effect would be to replace `\csname` by `\r@foo`; since we have three, the effect is to replace it by the value of `\r@foo`. There is a little trick. If the command is undefined, `\csname` sets `\r@foo` to `\relax`, and `\@secondoffive` produces `\relax`. In some cases, the command might produce some junk. We can avoid the 'missing number' error by inserting a zero in front of the token list. We get rid of the junk by evaluating everything else in a `\hbox`, that is copied in `\box0`, and discarded. Thus, this converts our page number in an integer, stored in `\sortcount`.

Our second job is then to sort. What we do is: we define `\sortoks` to be the empty token list, with an `\expandafter` that puts the content of the token list in the input stream. This means that we evaluate this token list in a context where the list is empty. Evaluating the list may have as side effect to insert this new element and replace the value by `\maxdimen`. Otherwise, we must insert it at the end.

```
\def\fosortpagecitation{%
   \setbox0\hbox{\global\sortcount=0\expandafter\expandafter\expandafter
   \@secondoffive\csname r@\FOrefid\endcsname
      \relax\relax\relax\relax\relax}
    \let\@elt\fosort@elt
    \global\sorttoks\expandafter{\expandafter}\the\sorttoks
    \ifnum\sortcount<\maxdimen
   \global\sorttoks\expandafter{%
       \the\expandafter\sorttoks\expandafter\@elt\expandafter{\the\sortcount}}
  \fi}
```

Let's compress the list. Consider the case where we have a list of numbers, say 1, 2, 3, 6, 8, and 9. We apply the next command to every element, in order. We assume that our numbers are positive, the counter is initialized to -2, so that the initial test is false. This means that the first number, 1, is typeset and remembered. The original code was wrong. We give here a correct version.

```
\gdef\focompress@elt#1{%
 \global\advance\sortcount\@ne
 \ifnum#1=\sortcount
    \def\foheld{\textendash#1}%
  \else
    \foheld\fosep#1\let\foheld\relax
 \fi
 \global\sortcount#1
 \def\fosep{, }}
```

This converts a XSLT format into a LaTeX command. There are examples in chapter 7 of this feature. It can be used to convert a section reference (defined by some numbers) into something like IV.7-2. If you say format=‘`[1.a]`’, and if the numbers are 4 and 2, the result is ‘`[4.b]`’. If we have only one number, the result is ‘`[4]`’. This mechanism is very complicated. We need only to typeset pages numbers (everything else is done by the XSLT processor), so that only one number has to be converted.

```
\expandafter\let\csname Format-1\endcsname\@arabic
\expandafter\let\csname Format-i\endcsname\@roman
\expandafter\let\csname Format-I\endcsname\@Roman
\expandafter\let\csname Format-a\endcsname\@alph
\expandafter\let\csname Format-A\endcsname\@Alph
```

Handling of `<fo:page-number/>`. We call a command that formats the page number.

```
\XMLelement{fo:page-number}
  {}
  {}
  {\expandafter\FOgeneratePage\FOformat\@null}
```

The call to the `\FOgeneratePage` command is a bit strange. The arguments `#1#2` is the result of the expansion of the format. Hence, `#1` is the first character of the format. This piece of code works if the format has the form ‘`1.`’: it applies `\@arabic` to the page number and puts a dot after it. For later use, we put the interesting part in a command.

```
\def\FOgeneratePage#1#2\@null{\csname Format-#1\endcsname{\c@page}#2}
\def\jgFOlabel#1{\csname Format-#1\endcsname{\c@page}}
```

We define now `\FOlabel`. The idea of the command is to create a label associated to the current id, stored in `\FOid`, provided that this is not empty. We do not execute the `\label` command, but instead, we write directly something in the .aux file. As explained above, the list contains five items, one of them is the page number (the first should be `\@currentlabel`, but this is unused, hence left empty). The hyperref package uses argument 4, that should be a unique ID, here we use id.

The implementation was modified. Originally, the quantity printed on the .aux file was the same as the body of `<fo:page-number>`. The trouble is that all quantities in the `\write` command are expanded; thus `\@arabic` is expanded; its value is `\number`, so that the current page number (from `\c@page`) is used. This is wrong. The `\protected@write` command redefines temporarily `\thepage` to be `\relax`, so that, in the case of `\label`, the page number is not evaluated (it will be later, when the page is shipped out). We corrected this in the following way. We add a local redefinition of `\jgFOlabel` to `\relax`. As a consequence, the `\write` will store the command and the value of the format attribute, the command will be expanded later. There is no need to this `\@null` hacking.

```
\def\FOlabel{%
 \ifx\@empty\FOid\else
  \@bsphack
  \protected@write\@mainaux{\let\jgFOlabel\relax}%
    {\string\newlabel{\FOid}{{}{\jgFOlabel\FOformat}{}{\FOid}{}}}%
  \@esphack
  \hyper@@anchor{\FOid}{\relax}%
  \global\let\FOid\@empty
 \fi
}
```

4.12.9 Other elements

The fotex.sty file defines a command `\@declaredcolor` that is unused. It defines this one:

```
\def\HTMLColor#1#2#3#4#5#6#7#8{%
 \definecolor{#1}{RGB}{"#3#4, "#5#6, "#7#8}}
```

This predefines some colors.

```
\HTMLColor{aqua}.00FFFF
\HTMLColor{black}.000000
\HTMLColor{blue}.0000FF
\HTMLColor{fuchsia}.FF00FF
\HTMLColor{gray}.808080
\HTMLColor{green}.008000
\HTMLColor{lime}.00FF00
\HTMLColor{maroon}.800000
\HTMLColor{navy}.000080
\HTMLColor{olive}.808000
\HTMLColor{purple}.800080
\HTMLColor{red}.FF0000
\HTMLColor{silver}.C0C0C0
\HTMLColor{teal}.008080
\HTMLColor{white}.FFFFFF
```

```
\HTMLColor{yellow}.FFFF00
\definecolor{orange}{cmyk}{0,0.61,0.87,0}
```

Leaders: This might produce – – – – – – – – – ..... ——————— in the case where the style is dashed, or dotted, or if the thickness is 1pt. If the width is zero, we emit the leader command, otherwise put it in a hbox of the desired width.

```
\XMLelement{fo:leader}
  {}
  {
 \leavevmode
  \ifx\FOleaderpattern\leader@pattern@rule
   \ifx\FOrulestyle\rule@style@dashed
    \def\w@t{\cleaders\hbox{$\m@th \mkern1.5mu-\mkern1.5mu$}\hfil}%
   \else
    \ifx\FOrulestyle\rule@style@dotted
      \def\w@t{\cleaders\hbox{$\m@th \mkern1.5mu.\mkern1.5mu$}\hfil}%
    \else
      \ifdim\FOrulethickness>\z@
        \def\w@t{\leaders\hrule height \FOrulethickness\hfill}%
      \else
        \def\w@t{\hfill}%
      \fi
    \fi
   \fi
  \else
  \ifx\FOleaderpattern\leader@pattern@dots
      \def\w@t{\cleaders\hbox{$\m@th \mkern1.5mu.\mkern1.5mu$}\hfill}%
  \else % space
      \def\w@t{\hfill}%
  \fi
 \fi
 \PercentToDimen{\FOleaderlength}%
 \ifdim\@tempdima=\z@\w@t\else\hbox to \@tempdima{\w@t}\fi
   }
  {}
```

Attributes and commands for handling leaders.

```
\XMLNSAX{fo}{leader-alignment}{\FOleaderalignment}{\inherit}
\XMLNSAX{fo}{leader-length}{\FOleaderlength}{\inherit}
\XMLNSAX{fo}{leader-pattern}{\FOleaderpattern}{\inherit}
\XMLNSAX{fo}{leader-pattern-width}{\FOleaderpatternwidth}{\inherit}
\XMLNSAX{fo}{rule-style}{\FOrulestyle}{\inherit}
\XMLNSAX{fo}{rule-thickness}{\FOrulethickness}{\inherit}
\gdef\FOleaderalignment{none}
\gdef\FOleaderlength{\z@}
\gdef\FOleaderpattern{space}
\gdef\FOleaderpatternwidth{}
\gdef\FOrulestyle{solid}
\gdef\FOrulethickness{1.0pt}
\XMLstringX\rule@style@dashed<>dashed</>
\XMLstringX\rule@style@dotted<>dotted</>
\XMLstringX\leader@pattern@space<>space</>
```

```
\XMLstringX\leader@pattern@rule<>rule</>
\XMLstringX\leader@pattern@dots<>dots</>
```

These are trivial.

```
\XMLelement{fo:block-container}
  {}  {}   {}
\XMLelement{fo:inline-container}
  {}  {}   {}
\XMLelement{fo:wrapper}
  {}
  {\FOSetFont{wrapper}\FOlabel}
  {}
```

These are undefined.

```
\XMLelement{fo:bidi-override}
\XMLelement{fo:initial-property-set}
\XMLelement{fo:instream-foreign-object}
\XMLelement{fo:multi-case}
\XMLelement{fo:multi-properties}
\XMLelement{fo:multi-property-set}
\XMLelement{fo:multi-switch}
\XMLelement{fo:multi-toggle}
\XMLelement{fo:table-footer}
```

Handling of `<fotex:bookmark FB-level=`*A* `FB-label=`*B*`>`*text*`</fotex:bookmark>`. We have written FB instead of 'fotex-bookmark', it's shorter. The translation is `\pdfbookmark[A]{text}{B}`.

```
\XMLelement{fotex:bookmark}
 {
  \XMLattributeX{fotex-bookmark-level}{\FOTEXbookmarklevel}{0}
  \XMLattributeX{fotex-bookmark-label}{\FOTEXbookmarklabel}{}
 }
 {\xmlgrab}
 {\protectCS\FOTEXbookmarklabel
   \let\ignorespaces\@empty
   \pdfbookmark[\FOTEXbookmarklevel]{#1}{\FOTEXbookmarklabel}}
```

What is the purpose of this?

```
\let\@@ReadBookmarks\ReadBookmarks
\def\ReadBookmarks{{\let\InputIfFileExists\@input\@@ReadBookmarks}}
```

Implementation of `<fo:character character=`*C*`/>`. This is a bit strange. We use more or less the same method as `<fo:inline>`.

```
\XMLNSA{fo}{character}{\FOcharacter}{}
\XMLelement{fo:character}
  {}
  {
  \ifx\FOverticalalign\att@auto \let\FOverticalalign\FObaselineshift \fi
  \FO@character{\FOcharacter}}
  {}
```

What we do is look at the vertical alignment attribute, and use some more or less standard functions for vertical placement.

```
\def\FO@character#1{%
 \ifx\FOverticalalign\att@baseline #1%
```

```
 \else \ifx\FOverticalalign\att@super \textsuperscript{#1}%
 \else \ifx\FOverticalalign\att@sub  \textsubscript{#1}%
 \else \PlayWithShift \raisebox{\dimen@}{#1}%
 \fi \fi \fi}
```

The `\textsuperscript` command is standard, this is not. We use the same idea.

```
\DeclareRobustCommand*\textsubscript[1]{%
  \@textsubscript{\selectfont#1}}
\def\@textsubscript#1{%
  {\m@th\ensuremath{_{\mbox{\fontsize\sf@size\z@#1}}}}}
```

4.13 Bootstrap code

Some integers, boxes and dimensions.

```
\newcount\FOTableNesting  \FOTableNesting0  % unused...
\newcount\FOinList        \FOinList0
\newdimen\FOspaceleft
\newdimen\MasterBottomMargin     \MasterBottomMargin\z@
\newdimen\MasterLeftMargin       \MasterLeftMargin\z@
\newdimen\MasterRightMargin      \MasterRightMargin\z@
\newdimen\MasterTopMargin        \MasterTopMargin\z@
\newdimen\XFOendindent  \newdimen\XFOstartindent % unused
\newdimen\bottommargin
\newdimen\FOtempdim
\newdimen\@default \@default=10pt
\newsavebox\BlockBox
\newsavebox\CellBox
\newsavebox\FOBOX
```

Some conditionals.

```
\newif\ifFOBlockGrab   \FOBlockGrabfalse
\newif\ifFODebug       \FODebugfalse
\newif\ifFOListBody    \FOListBodyfalse
\newif\ifFOListInnerPar\FOListInnerParfalse
\newif\ifFOinOutput    \FOinOutputfalse
```

```
\def\DEBUG#1{%
 \ifFODebug \typeout{#1, at \the\inputlineno} \fi
}
```

These are defined, but currently ignored.

```
\def\usewhitespace{%
 \UnicodeCharacter{13}{ \ignorespaces}%
 \UnicodeCharacter{32}{ \ignorespaces}%
 \UnicodeCharacter{9}{ \ignorespaces}%
}
\def\ignorewhitespace{%
 \UnicodeCharacter{13}{}%
 \UnicodeCharacter{32}{}%
 \UnicodeCharacter{9}{}%
}
```

Some strings, used everywhere.

```
\XMLstring\LINK<>LINK</>
\XMLstringX\att@all<>all</>
\XMLstringX\att@any<>any</>
\XMLstringX\att@auto<>auto</>
\XMLstringX\att@blank<>blank</>
\XMLstringX\att@black<>black</>
\XMLstringX\att@bottom<>bottom</>
\XMLstringX\att@centered<>center</>
\XMLstringX\att@false<>false</>
\XMLstringX\att@first<>first</>
\XMLstringX\att@no<>no</>
\XMLstringX\att@none<>none</>
\XMLstringX\att@normal<>normal</>
\XMLstringX\att@page<>page</>
\XMLstringX\att@pre<>pre</>
\XMLstringX\att@repeat<>repeat</>
\XMLstringX\att@true<>true</>
```

Some attributes.

```
\XMLNSA{fo}{color}{\FOcolor}{\inherit}    \gdef\FOcolor{}
\XMLNSAX{fo}{id}{\FOid}{}
\XMLNSAX{fo}{role}{\FOrole}{none}
\XMLNSAX{fo}{size}{\FOsize}{auto}
\XMLNSAX{fo}{src}{\FOsrc}{}
\XMLNSAX{fo}{text-indent}{\FOtextindent}{\inherit} \gdef\FOtextindent{\z@}
\XMLNSAX{fo}{top}{\FOtop}{auto}
\XMLNSA{fo}{vertical-align}{\FOverticalalign}{auto}
\XMLNSAX{fo}{width}{\FOwidth}{auto}
\XMLNSAX{fotex}{column-align}{\FOcolumnalign}{}
\XMLNSAX{fo}{format}{\FOformat}{\inherit}    \gdef\FOformat{1}
```

Some names.

```
\XMLname{fo:list-item-label}{\FOListItemLabel}
\XMLname{fo:list-item-body}{\FOListItemBody}
\XMLname{fo:table-cell}{\FOTableCell}
\XMLname{fo:table-row}{\FOTableRow}
```

This is executed when we load the file. Comments of the form D=foo indicate the default value in LaTeX.

```
\def\fps@table{!htbp} %D=tbp
\def\fps@figure{!htbp} %D=tbp
\parindent\z@      %D=20pt
\parskip\z@        %D=0pt plus 1pt
\emergencystretch 3em  %D=0.0pt
\tabcolsep3pt      %D=6pt
\hbadness=4000     %D=1000
\hyphenpenalty=400 %D=50
\pretolerance=500  %D=100
\relpenalty=500
\tolerance=1000    %D=200
\vbadness=3000     %D=1000
\widowpenalty=8000 %D=150
\clubpenalty=8000  %D=150
```

```
\@twosidetrue
\fboxsep0pt        %D=3pt
\setcounter{topnumber}{5}        %D=2
\renewcommand\topfraction{.9}    %D=.7
\setcounter{bottomnumber}{12}    %D=1
\renewcommand\bottomfraction{.9} %D=.3
\setcounter{totalnumber}{6}      %D=3
\renewcommand\textfraction{.1}   %D=.2

\DefineCharacter{8232}{2028}{\newline}
\DefineCharacter{8208}{2010}{-\/}
\@ifundefined{pdfoutput}{}{\def\pdfBorderAttrs{/Border [0 0 0]}}
\long\def\@firstoffive#1#2#3#4#5{#1}%
\long\def\@secondoffive#1#2#3#4#5{#2}%
\long\def\@thirdoffive#1#2#3#4#5{#3}%
\long\def\@fourthoffive#1#2#3#4#5{#4}%
\long\def\@fifthoffive#1#2#3#4#5{#5}%

\def\supppdf{supp-pdf}
\let\FOinputIfFileExists\InputIfFileExists
\def\InputIfFileExists#1#2#3{%
 {\def\@tempa{#1}\ifx\@tempa\supppdf\else
   \FOinputIfFileExists{#1}{#2}{#3}\fi}}
\providecommand\textasciitilde{~}
```

Chapter 5

Converting XML to XML

This chapter, and the next one, describes three types of style sheets: they convert XML to XML, to XSL/Format and HTML. Originally, in 2002 and 2003, the XML files created by Tralics were used directly for production of the HTML and Pdf version , but in 2004, a new DTD was designed for the Raweb. The name of this new DTD was unclear for a long time; it is now 'raweb2.dtd' and the old name is 'raweb3.dtd' (the 3 here is for 2003). We shall explain here the style sheets that convert from the old DTD to the new one, and from this to HTML; we shall also explain the style sheets that convert from the old DTD to XSL/Format (those for the new one are similar).

The style sheets for converting into XSL/Format are adaptations by José Grimm of the TEI code (by Sebastian Rahtz). These are part of the Tralics distribution. Other files were written by J. Grimm (conversion to HTML) or Tahia Benhaj Abdellatif (conversion to XML) and maintained by Marie-Pierre Durollet and Bruno Marmol. The Tralics files have a Copyright notice that looks like this:

```
<!-- Copyright Inria 2003-2004 Jose Grimm. This file is an adaptation of
files from the TEI distribution. See original Copyright notice below.
-->
```

The "original Copyright notice" is given here:

```
<!--
 Copyright 1999-2001 Sebastian Rahtz/Oxford University
    <sebastian.rahtz@oucs.ox.ac.uk>

 Permission is hereby granted, free of charge, to any person obtaining
 a copy of this software and any associated documentation files (the
 ''Software''), to deal in the Software without restriction, including
 without limitation the rights to use, copy, modify, merge, publish,
 distribute, sublicense, and/or sell copies of the Software, and to
 permit persons to whom the Software is furnished to do so, subject to
 the following conditions:

 The above copyright notice and this permission notice shall be included
 in all copies or substantial portions of the Software.
-->
```

Let's consider an example. This is the start of a document created by Tralics:

```
<?xml version='1.0' encoding='iso-8859-1'?>
<!DOCTYPE raweb SYSTEM 'raweb3.dtd'>
<!-- translated from latex by tralics 2.4-->
```

```
<raweb language='english' creator='Tralics version2.4' year='2004'>
<accueil isproject='false' html='apics'>
```

This is the start of the translation to the new DTD:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!--translated from old xml 2003 by with 2XMLvalideDTD2.xsl-->
<!DOCTYPE raweb PUBLIC "-//INRIA//DTD Raweb 2" "raweb2.dtd">
<raweb xmlns:html="http://www.w3.org/1999/xhtml" xml:lang="en" year="2004">
  <identification isproject="false" id="apics">
    <shortname>apics</shortname>
```

There is a second style sheet that adds ids to some elements. The resulting file starts like this:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE raweb SYSTEM "raweb.dtd">
<raweb xmlns:html="http://www.w3.org/1999/xhtml"
       xmlns:xlink="http://www.w3.org/1999/xlink" id="id2243496"
       xml:lang="en" year="2004">
  <identification id="apics" isproject="false">
    <shortname id="id2267539">apics</shortname>
```

Note the following details: Tralics uses simple quotes when it outputs its tree; the XSLT processor used by the Raweb uses double quotes; in the style files random values are used. In this exemple the output of the XSLT processor is 'indented', this is an option that depends on the style sheet. Tralics never indents.

5.1 Converting the XML to the new DTD

All files start like this, we shall not repeat this line.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
```

The root element here is `<xsl:transform>`; in some other files, it can be `<xsl:stylesheet>`, this is the same. We declare the namespaces. The 'html' namespace is not used here.

```
<xsl:transform
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
        xmlns:xlink="http://www.w3.org/1999/xlink"
        xmlns:html="http://www.w3.org/1999/xhtml"
        exclude-result-prefixes="xlink">
```

We use an auxiliary file for the bibliography; this will be defined in the next section.

```
<xsl:import href="2XMLvalideDTD2-biblio.xsl"/>
```

The style sheet contains this line, but it is useless since we will use `<xsl:document>`.

```
<xsl:output method='xml' doctype-system='raweb2.dtd' indent='yes'
    encoding='iso-8859-1'/>
```

Spaces are removed in some elements, listed here. The list contains a single name, `<UR>`.

```
<xsl:strip-space elements="UR"/>
```

The root of the Raweb is the `<raweb>` element. Its children are `<accueil>` (that gives some information about the team; this is used to produce a title page[1]), `<moreinfo>` (this is optional, it is a short piece of text, annexed to a section, or the whole document), `<composition>` (this describes the members of the team), `<presentation>`, `<fondements>`[2], `<domaine>`, `<logiciels>`[3],

[1] In French, accueil means reception, welcome.
[2] Fondements = foundations
[3] Logiciel = software

<resultats>, <contrats>, <international>, <diffusion> (these eight elements are called 'sections', they have the same structure and contain text, that can be divided into subsections, etc.), and <biblio> (this is the bibliography).

The table of contents of the Raweb has ten entries: Members, Overall Objectives, Scientific Foundations, Application Domains, Software, New Results, Contracts and Grants with Industry, Other Grants and Activities, Dissemination, Bibliography. Each entry comes from one of these elements, except <accueil> and <moreinfo>, that play a special role. In particular the html attribute of <accueil> is the Team's name in lower case ASCII 7bits. This is also the name of the source file, and a prefix for output files. We store in $LeProjet this name[4].

```
<xsl:variable name="LeProjet" select="/raweb/accueil/@html"/>
```

The <raweb> element has two important attributes, language and year. We put in the variable $year this quantity with a default value of 2004.

```
<xsl:variable name="year">
 <xsl:choose>
  <xsl:when test='/raweb/@year'><xsl:value-of select="/raweb/@year"/></xsl:when>
  <xsl:otherwise>2004</xsl:otherwise>
 </xsl:choose>
</xsl:variable>
```

This is the main rule. The translation of <raweb> is the file apics-dtd2.xml (assuming that $LeProjet is 'apics'), its root element is <raweb>. This element has some attributes, namely namespaces, year (from the $year variable) and xml:lang from the language attribute. The content is formed by the transformation of the eight standard sections, followed by the bibliography, preceded by the transformation of <accueil> and <composition>.

```
<xsl:template match="/raweb">
 <xsl:document href="{$LeProjet}-dtd2.xml" method="xml"
    doctype-public="-//INRIA//DTD Raweb 2"  doctype-system="raweb2.dtd"
    indent='yes' encoding='iso-8859-1'>
 <xsl:comment>translated from old xml 2003 by with 2XMLvalideDTD2.xsl</xsl:comment>
 <raweb
       xmlns:xlink="http://www.w3.org/1999/xlink"
       xmlns:html="http://www.w3.org/1999/xhtml">
   <xsl:attribute name="lang" namespace="http://www.w3.org/XML/1998/namespace">
      <xsl:choose>
         <xsl:when test="@language='english'">en</xsl:when>
         <xsl:when test="@language='french'">fr</xsl:when>
         <xsl:otherwise><xsl:value-of select="@language"/></xsl:otherwise>
      </xsl:choose>
   </xsl:attribute>
   <xsl:attribute name="year"><xsl:value-of select="{$year}"/></xsl:attribute>
   <xsl:apply-templates select="accueil"/>
   <xsl:call-template name="topic"/>
   <xsl:apply-templates select="presentation |
                        fondements | domaine | logiciels | resultats |
                        contrats | international | diffusion"/>
   <xsl:apply-templates select="biblio"/>
  </raweb>
 </xsl:document>
</xsl:template>
```

A topic is an attribute of a module. Modules can be ordered by section or by topic; for this reason there are two style sheets that convert the XML into HTML. The table of contents of the HTML version thas a button that switches form one view to the other; there is only one style sheet

[4] 'LeProjet' is French for 'TheTeam'.

for the Pdf, it ignores these topic attributes. A topic attribute is a reference to a topic declaration like `<topic num='1'><t_titre> High-level modeling</t_titre></topic>`. The value of the num attribute is computed by Tralics; specifications say that it should be an integer. In the new DTD, it should be an ID, so that we transform it to `t_1` (the IDs generated by Tralics are of the form uid125 or bid125, so that this transformation does not conflict with already existing IDs). The `<t_titre>` element is useless here; it was removed in the new DTD. The topic declarations are moved from inside `<accueil>` to after `<identification>`.

```
<xsl:template name="topic">
 <xsl:for-each select="accueil/child::topic">
  <xsl:element name="topic">
   <xsl:attribute name="id">t_<xsl:value-of select="@num"/></xsl:attribute>
     <xsl:value-of select="./t_titre"/>
   </xsl:element>
 </xsl:for-each>
/xsl:template>
```

This piece of code is a copy from the TEI. The idea is to leave math formulas unchanged. It will be used in all style sheets.

```
<xsl:template match="*|@*|comment()|processing-instruction()|text()" mode="math">
 <xsl:copy>
  <xsl:apply-templates mode="math" select="*|@*|processing-instruction()|text()"/>
 </xsl:copy>
</xsl:template>
```

A `<formula>` is a wrapper for a `<math>` expression. The action here is to copy the attributes and the content.

```
<xsl:template match="formula">
    <xsl:element name="formula">
       <xsl:copy-of select="@*"/><xsl:apply-templates mode="math"/>
    </xsl:element>
</xsl:template>
```

If a formula has the attribute type=‘display’, it is a display math formula, outside any paragraph. We put it in a `<p>` element.

```
<xsl:template match="formula[@type='display']">
 <p>
  <xsl:element name="formula">
     <xsl:copy-of select="@*"/><xsl:apply-templates mode="math"/>
  </xsl:element>
 </p>
</xsl:template>
```

We convert `<accueil>` into `<identification>`. We copy the isproject attribute (this is ‘true’ in the case where the team is a project, ‘false’ otherwise). We copy the html trait, renaming it id. After that, we add the transformations of the following elements: `<projet>`, `<projetdeveloppe>`, `<theme>`, `<composition>` (this is a sibling), `<UR>`, and `<moreinfo>` (this is optional).

```
<xsl:template match="accueil">
 <xsl:element name="identification">
  <xsl:attribute name="isproject"><xsl:value-of select="@isproject"/></xsl:attribute>
  <xsl:attribute name="id"><xsl:value-of select="@html"/></xsl:attribute>
  <xsl:apply-templates select="projet"/>
  <xsl:apply-templates select="projetdeveloppe"/>
  <xsl:apply-templates select="theme"/>
  <xsl:apply-templates select="../composition"/>
  <xsl:apply-templates select="UR"/>
  <xsl:if test="/raweb/moreinfo">
    <xsl:apply-templates select="/raweb/moreinfo"/>
```

```
  </xsl:if>
 </xsl:element>
</xsl:template>
```

We copy the <theme> element, replacing lowercase letters by uppercase ones.

```
<xsl:template match="accueil/theme">
 <xsl:element name="theme">
   <xsl:value-of select="translate(.,'abcdefghijklmnopqrstuvwxyz',
                                    'ABCDEFGHIJKLMNOPQRSTUVWXYZ')"/>
 </xsl:element>
</xsl:template>
```

We copy the <projet> element, renaming it <shortname>. Note that, for the LaTeX team, this could be <LaTeX>, so that a simple copy is not enough.

```
<xsl:template match="accueil/projet">
 <xsl:element name="shortname"> <xsl:apply-templates />  </xsl:element>
</xsl:template>
```

We copy the <projetdeveloppe> element, renaming it <projectName>[5]. Note that this element can have font changes, hence we must process the content.

```
<xsl:template match="accueil/projetdeveloppe">
 <xsl:element name="projectName"> <xsl:apply-templates /> </xsl:element>
</xsl:template>
```

In the case of <UR>, we consider only the content; it should be a sequence of elements of the form <URxxx>, these elements are listed below.

```
<xsl:template match="UR">
   <xsl:apply-templates />
</xsl:template>
```

We replace <URRocquencourt>, <URRhoneAlpes>, <URRennes>, <URLorraine>, <URFuturs>, and <URSophia> by <UR name='Rocquencourt'/>, etc. These elements are empty, they represent one of the six INRIA's research units.

```
<xsl:template match="URRocquencourt">
   <UR name="Rocquencourt" />
</xsl:template>
<xsl:template match="URRhoneAlpes">
   <UR name="RhoneAlpes"/>
</xsl:template>
<xsl:template match="URRennes">
   <UR name="Rennes"/>
</xsl:template>
<xsl:template match="URLorraine">
   <UR name="Lorraine"/>
</xsl:template>
<xsl:template match="URFuturs">
   <UR name="Futurs" />
</xsl:template>
<xsl:template match="URSophia">
   <UR name="Sophia" />
</xsl:template>
```

Translation of <presentation>, <fondements>, <domaine>, <logiciels>, <resultats>, <contrats>, <international>, and <diffusion>. We simplified a bit the code by assuming that the id attribute is present (otherwise, the code is wrong). The result is an element of the same name. The only difference is that we convert the titre attribute in a <bodyTitle> element. This

[5]Here the French word développé is used in the sense of expanded. All tag names are ASCII 7bits, although XML allows any Unicode character.

attribute is constant, defined in the DTD, see page 241 and following, lines 181, 186, 191, 196, 201, 206, 211, 216, and 221.

```
<xsl:template match="presentation | fondements | domaine | logiciels |
    resultats | contrats | international | diffusion">
 <xsl:variable name="nodename" select="name()"/>
 <xsl:element name="{$nodename}">
   <xsl:attribute name="id">  <xsl:value-of select="@id"/>  </xsl:attribute>
   <xsl:element name="bodyTitle">
       <xsl:value-of select="@titre"/>
   </xsl:element>
   <xsl:apply-templates/>
 </xsl:element>
</xsl:template>
```

Translation of <module>. The result is a <subsection>. We convert, in order, <head> (this is the title of the module), <participant>, <participante>, <participants>, <participantes> (four variants that indicate the participants to the action described in the module), <keywords> (the keywords), <moreinfo> (the 'moreinfo' data structure; we grab the first element, its transformation will read the other siblings), and finally everything else. A module has two attributes id and topic that are copied. If the topic is present, we have to convert the value as above line 46 ('12' replaced by 't_12').

```
<xsl:template match="module">
 <xsl:element name="subsection">
   <xsl:if test="@topic and @topic!=''">
     <xsl:attribute name="topic">t_<xsl:value-of select="@topic"/>
     </xsl:attribute>
   </xsl:if>
   <xsl:call-template name="id"/>
   <xsl:apply-templates select="head" mode="caption"/>
   <xsl:apply-templates
      select="participants | participant | participantes | participante"/>
   <xsl:apply-templates select="keywords"/>
   <xsl:apply-templates select="moreinfo[position()=1]"/>
   <xsl:apply-templates select="node()[local-name() != 'moreinfo'
       and local-name()!='keywords' and local-name()!='head'
       and local-name()!='participants' and local-name()!='participant'
       and local-name()!='participante' and local-name()!='participantes' ]"/>
 </xsl:element>
</xsl:template>
```

Translation of <div0>, <div1>, <div2>, <div3>, and <div4>. The result is a <subsection>, the code is the same as for a module, except that these elements have no topic attribute.

```
<xsl:template match="div0 | div1 | div2 | div3 | div4">
 <xsl:element name="subsection">
   <xsl:call-template name="id"/>
   <xsl:apply-templates select="head" mode="caption"/>
   <xsl:apply-templates
        select="participants | participant | participantes | participante"/>
   <xsl:apply-templates select="keywords"/>
   <xsl:apply-templates select="moreinfo[position()=1]"/>
   <xsl:apply-templates select="node()[...]" />         <!-- as above l. 139-142-->
 </xsl:element>
</xsl:template>
```

Transformation of <moreinfo>. The result is a <moreinfo> that contains the content of the element and all the following siblings.

```
155 <xsl:template match="moreinfo">
156  <xsl:element name="moreinfo">
157    <xsl:apply-templates/>
158    <xsl:for-each select="following-sibling::moreinfo">
159      <xsl:apply-templates/>
160    </xsl:for-each>
161  </xsl:element>
162 </xsl:template>
```

Transformation of <composition>. The result is a <team> element, containing the <catperso> children and an optional <moreinfo> (let's hope there is only one, because of the code line 158).

```
163 <xsl:template match="composition">
164  <xsl:element name="team">
165    <xsl:call-template name="id"/>
166    <xsl:call-template name="catperso"/>
167    <xsl:apply-templates select="moreinfo"/>
168  </xsl:element>
169 </xsl:template>
```

The transformation of <catperso><head>foo</head>etc</catperso> is a <participants> element with an attribute category='foo', with spaces replaced by underscores, and whose content is the translation of all <pers> elements it contains (the semantics is: a <catperso> contains a title in <head>, that could be 'Ph.D. Students', followed by some <pers> elements, all the students of the team).

```
170 <xsl:template name="catperso">
171  <xsl:for-each select="catperso">
172    <xsl:element name="participants">
173      <xsl:attribute name="category">
174        <xsl:value-of select="translate(./head, ' ', '_')"/>
175      </xsl:attribute>
176      <xsl:apply-templates select="pers"/>
177    </xsl:element>
178  </xsl:for-each>
179 </xsl:template>
```

The transformation of <participants> is also a <participants> element, where the category attribute has value 'None'. Originally, we had four elements, this one and <participant>, <participante>, <participantes>. This was simplified: the difference between masculine and feminine does not appear in English; the final s is removed, it will be added later if the list contains more than one element.

```
180 <xsl:template match="participants | participant | participantes | participante">
181   <xsl:element name="participants">
182     <xsl:attribute name="category">None</xsl:attribute>
183     <xsl:apply-templates/>
184   </xsl:element>
185 </xsl:template>
```

The transformation of <pers prenom='Donald' nom='Knuth'>Author of <TeX/> </pers> is a <person> element, with three children, the first is <firstname>, the second is <lastname>, they contain the prenom and nom, and the last one is a <moreinfo> element that contains the content of this element; it is optional. The test is strange because later on, lines 1031 and 1045, we test again for emptyness, but white space is normalised there, not here.

```
186 <xsl:template match="pers">
187  <xsl:element name="person">
188    <xsl:call-template name="id"/>
189    <xsl:element name="firstname"><xsl:value-of select="./@prenom"/></xsl:element>
190    <xsl:element name="lastname"><xsl:value-of select="./@nom"/> </xsl:element>
```

```
191     <xsl:if test="string-length(.) > 0">
192        <xsl:element name="moreinfo"> <xsl:apply-templates/> </xsl:element>
193     </xsl:if>
194   </xsl:element>
195 </xsl:template>
```

The element `<refperson>` is not defined in the old DTD. We can leave it unchanged.

```
196 <xsl:template match="refperson"> <xsl:copy-of select="."/> </xsl:template>
```

Transformation of `<hi rend=XX>text</hi>`. The result depends on the value of the attribute. If the attribute is 'sup', we construct a `<sup>` element; if the attribute is 'sub', we construct a `<sub>` element; if the attribute is 'bold', we construct a `<b>` element, with a hack: if you use the obsolete environments body and abstract, Tralics inserts a warning in the document, this is removed here[6]; if the attribute is 'small', we construct a `<small>` element; if the attribute is 'large', we construct a `<big>` element; if the attribute is 'tt', we construct a `<tt>` element; if the attribute is 'sc', we construct a `<span>` element[7]; if the attribute is 'center'[8], we construct a `<span>` element; if the attribute is 'underline' we construct a `<em>` element; otherwise, the result is a `<i>` element.

```
197 <xsl:template match="hi">
198   <xsl:choose>
199    <xsl:when test="@rend = 'sup'"> <sup><xsl:apply-templates/></sup></xsl:when>
200    <xsl:when test="@rend = 'sub'"> <sub><xsl:apply-templates/></sub></xsl:when>
201    <xsl:when test="@rend = 'bold'">
202     <xsl:if test=".!='Body (obsolete)'">
203        <xsl:if test=".!='Abstrat (obsolete)'">
204          <b><xsl:apply-templates/></b>
205        </xsl:if>
206     </xsl:if>
207    </xsl:when>
208    <xsl:when test="@rend = 'small'">
209      <small><xsl:apply-templates/></small> </xsl:when>
210    <xsl:when test="@rend = 'sc'">
211      <span class="smallcap"> <xsl:value-of select="."/> </span>
212    </xsl:when>
213    <xsl:when test="@rend = 'large'"><big><xsl:apply-templates/></big> </xsl:when>
214    <xsl:when test="@rend = 'center'">
215       <span align="center"><xsl:apply-templates/></span>
216    </xsl:when>
217    <xsl:when test="@rend = 'underline'">
218       <em style="UNDERLINE"><xsl:apply-templates/></em>
219    </xsl:when>
220    <xsl:when test="@rend = 'tt'"> <tt><xsl:apply-templates/></tt> </xsl:when>
221    <xsl:otherwise> <i><xsl:apply-templates/></i> </xsl:otherwise>
222   </xsl:choose>
223 </xsl:template>
```

Transformation of `<keywords>`. We consider only the `<term>` children, changing the name to `<keyword>` (there should be no other children).

```
224 <xsl:template match="keywords">
225    <xsl:for-each select="term">
226       <xsl:element name="keyword">
227          <xsl:call-template name="id"/>
228          <xsl:value-of select="."/>
229       </xsl:element>
```

[6] This should be removed for 2005. The error message changed.

[7] There is no apply-templates here; this is wrong (JG).

[8] A paragraph can be centered, as well as a cell in table, but not inline elements like `<hi>`.

```
230    </xsl:for-each>
231 </xsl:template>
```

Transformation of <code>. This is trivial.

```
232 <xsl:template match="code">
233    <xsl:element name="code"> <xsl:apply-templates/> </xsl:element>
234 </xsl:template>
```

Transformation of <ref>. The result is an element of the same name. It has the same id (does anybody reference a reference?). The target attribute is replaced by a xlink:href attribute, with the same value, but it has a # in front. There is also a location attribute whose value is 'intern', except when the parent is a <cit>, case where 'biblio' is used.

```
235 <xsl:template match="ref">
236    <xsl:element name="ref">
237       <xsl:call-template name="id"/>
238       <xsl:attribute name="xlink:href" namespace="http://www.w3.org/1999/xlink">
239          <xsl:value-of select="concat('#', @target)"/>
240       </xsl:attribute>
241       <xsl:attribute name="location">
242          <xsl:choose>
243             <xsl:when test="parent::cit">biblio</xsl:when>
244             <xsl:otherwise>intern</xsl:otherwise>
245          </xsl:choose>
246       </xsl:attribute>
247       <xsl:apply-templates/>
248    </xsl:element>
249 </xsl:template>
```

Transformation of <cit>. We transform only the content, which should be a single <ref>. It is possible to know that the <ref> comes from a <cit> because of its location attribute.

```
250 <xsl:template match="cit">  <xsl:apply-templates/> </xsl:template>
```

Transformation of <xref>. We simplified the code by removing a test that was wrong, hence always false. The result is the same as <ref>, but location is always external, and the link is unchanged.

```
251 <xsl:template match="xref">
252      <xsl:element name="ref">
253       <xsl:call-template name="id"/>
254       <xsl:attribute name="xlink:href" namespace="http://www.w3.org/1999/xlink">
255          <xsl:value-of select="@url"/>
256       </xsl:attribute>
257       <xsl:attribute name="location">extern</xsl:attribute>
258       <xsl:apply-templates/>
259      </xsl:element>
260 </xsl:template>
```

The <ident> element is unused. It should contain only text.

```
261 <xsl:template match="ident"> <xsl:copy/> </xsl:template>
```

Transformation of <note>; the result is <footnote>.

```
262 <xsl:template match="note">
263    <xsl:element name="footnote"><xsl:copy-of select="@*"/>
264       <xsl:call-template name="id"/>
265       <xsl:apply-templates/>
266    </xsl:element>
267 </xsl:template>
```

Transformation of <p>. Trivial. Note: we shall see later that there is a second rule for this element.

```
<xsl:template match="p">
   <xsl:element name="p">
      <xsl:copy-of select="@*" />
      <xsl:apply-templates/>
   </xsl:element>
</xsl:template>
```

Transformation of <list>. The result is <descriptionlist>, <glosslist>, <orderedlist>, <simplelist>, depending on the value of type attribute.

```
<xsl:template match="list">
  <xsl:choose>
   <xsl:when test="@type='description'">
     <xsl:element name="descriptionlist">
        <xsl:call-template name="id"/>  <xsl:apply-templates/>
     </xsl:element>
   </xsl:when>
   <xsl:when test="@type='gloss'">
     <xsl:element name="glosslist">
       <xsl:call-template name="id"/> <xsl:apply-templates/>
     </xsl:element>
   </xsl:when>
   <xsl:when test="@type='ordered'">
     <xsl:element name="orderedlist">
       <xsl:call-template name="id"/> <xsl:apply-templates/>
     </xsl:element>
   </xsl:when>
   <xsl:otherwise>
     <xsl:element name="simplelist">
       <xsl:call-template name="id"/> <xsl:apply-templates/>
     </xsl:element>
   </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Transformation of <item>: the result is <li>.

```
<xsl:template match="item">
   <xsl:element name="li">
      <xsl:call-template name="id"/> <xsl:apply-templates/>
   </xsl:element>
</xsl:template>
```

Transformation of <label>: the result is <label>.

```
<xsl:template match="label">
   <xsl:element name="label">
      <xsl:call-template name="id"/>  <xsl:apply-templates/>
   </xsl:element>
</xsl:template>
```

Transformation of <table>. The result is a <table>. We set the attribute border to 'solid' in case one of the cells in the table has a bottom-border attribute that is true.[9] The rend attribute is copied. We copy all <row> children, followed by the <caption>, if there is one (normally, there is none), followed by <head>, renamed to <caption>[10].

[9] This is strange; the test should be done on the rows, and all borders.

[10] This seems to be a bug. Tralics converts both 'table' and 'tabular' environments to <table>, in the first case, there is no caption.

```
<xsl:template match="table">
 <xsl:element name="table">
   <xsl:if test="./row/cell/@bottom-border='true'">
     <xsl:attribute name="border">solid</xsl:attribute>
   </xsl:if>
   <xsl:if test="./@rend">
    <xsl:attribute name="rend"><xsl:value-of select="./@rend" /></xsl:attribute>
   </xsl:if>
   <xsl:call-template name="id"/>
   <xsl:apply-templates select="row" />
   <xsl:apply-templates select="caption"/>
   <xsl:element name="caption"> <xsl:value-of select="head"/> </xsl:element>
  </xsl:element>
</xsl:template>
```

Transformation of `<row>`. The test here is strange. In the case where the test is false, the result is a `<tr>`, with the same content as the row. There is one attribute style[11] obtained from the right-border, top-border, left-border, and bottom-border attributes.

```
<xsl:template match="row">
  <xsl:choose>
    <xsl:when test="normalize-space(.) = '' and not(cell/child::*)">

    </xsl:when>
    <xsl:otherwise>
     <xsl:element name="tr">
       <xsl:attribute name="style">
        <xsl:if test="@right-border='true'"
                >border-right-style:solid;border-right-width:1px;</xsl:if>
        <xsl:if test="@top-border='true'"
                >border-top-style:solid;border-top-width:1px;</xsl:if>
        <xsl:if test="@left-border='true'"
                >border-left-style:solid;border-left-width:1px;</xsl:if>
        <xsl:if test="@bottom-border='true'"
                >border-bottom-style:solid; border-bottom-width:1px;</xsl:if>
       </xsl:attribute >
       <xsl:apply-templates/>
     </xsl:element>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

The transformatoin of `<cell>` is `<td>`, with the same content. There is one attribute style obtained from the halign, right-border, top-border, left-border, and bottom-border attributes. Attributes rows and cols are copied if the value is greater than one (this is the row span or column span of the cell).

```
<xsl:template match="cell">
 <xsl:element name="td">
   <xsl:attribute name="style">
    <xsl:if test="@halign"
       >text-align:<xsl:value-of select="@halign"/>;</xsl:if>
    <xsl:if test="@right-border='true'"
       >border-right-style:solid;border-right-width:1px;</xsl:if>
    <xsl:if test="@top-border='true'"
       >border-top-style:solid;border-top-width:1px;</xsl:if>
```

[11] The result is so complicated that the XSL/Format style sheet ignores this attribute. This should be changed in 2005.

```
353      <xsl:if test="@left-border='true'"
354         >border-left-style:solid;border-left-width:1px;</xsl:if>
355      <xsl:if test="@bottom-border='true'"
356         >border-bottom-style:solid; border-bottom-width:1px;</xsl:if>
357     </xsl:attribute >
358     <xsl:if test="./@cols>1">
359       <xsl:attribute name="cols"><xsl:value-of select="./@cols" /></xsl:attribute>
360     </xsl:if>
361     <xsl:if test="./@rows>1">
362       <xsl:attribute name="rows"><xsl:value-of select="./@rows" /></xsl:attribute>
363     </xsl:if>
364    <xsl:apply-templates/>
365   </xsl:element>
366 </xsl:template>
```

Attributes halign are always copied. This should be explained, because, a priori, all these attributes were converted to a style attribute.

```
367 <xsl:template match="@halign">
368    <xsl:attribute name="halign"><xsl:value-of select="." /></xsl:attribute>
369 </xsl:template>
```

This converts a <figure> element into a <ressource> element. The rend attribute is copied under the name type. Other attributes like width, height, scale, angle, and framed are just copied.

```
370 <xsl:template name="ressource">
371   <ressource xlink:href="{@file}">
372     <xsl:if test="@rend">
373       <xsl:attribute name="type"><xsl:value-of select="@rend"/></xsl:attribute>
374     </xsl:if>
375     <xsl:if test="@width"> <xsl:copy-of select="@width"/> </xsl:if>
376     <xsl:if test="@height"> <xsl:copy-of select="@height"/> </xsl:if>
377     <xsl:if test="@scale">  <xsl:copy-of select="@scale"/> </xsl:if>
378     <xsl:if test="@angle"> <xsl:copy-of select="@angle"/>  </xsl:if>
379     <xsl:if test="@framed"> <xsl:copy-of select="@framed"/> </xsl:if>
380     <xsl:if test="head and ((ancestor::figure) or not(@file))">
381       <xsl:apply-templates select="head"  mode="caption"/>
382     </xsl:if>
383   </ressource>
384 </xsl:template>
```

In the case where a <figure> is in a <table> which is in a <figure>, and if it has a file attribute, then the result is a 'ressource'.

```
385 <xsl:template match="figure//table//figure[@file]" priority="5">
386   <xsl:call-template name="ressource"/>
387 </xsl:template>
```

In the case where a <figure> has a file attribute, is below a figure, but does not match the rule above, then the result is a 'ressource', as above, but in a <td>.

```
388 <xsl:template match="figure[(ancestor::figure) and @file]">
389   <td><xsl:call-template name="ressource"/></td>
390 </xsl:template>
```

This is the last rule for a <table>. Let's hope no case is forgotten. The result is a <object>. It contains a <table>: in the case where there is a file attribute, the element is empty, and the translation is a <table> with a single <tr> with a single <td> with the ressource. In the case where there is a <p> with a table, we consider only these elements (let's hope for the best). Otherwise, we add a <table>, and each <p> will produce a row. A caption is put at the end.

```
<xsl:template match="figure[not(ancestor::figure)]">
   <xsl:element name="object">
      <xsl:call-template name="id"/>
      <xsl:choose>
         <xsl:when test="@file">
            <table>
              <tr><td>
              <xsl:call-template name="ressource"/>
              </td></tr>
            </table>
         </xsl:when>
         <xsl:when test="p/table">
            <xsl:apply-templates select="p/table" />
         </xsl:when>
         <xsl:otherwise>
            <table> <xsl:apply-templates /> </table>
         </xsl:otherwise>
      </xsl:choose>
      <xsl:apply-templates select="head"  mode="caption"/>
   </xsl:element>
</xsl:template>
```

This code is applied only in the 'otherwise' case of the previous template. For 2005, the best thing to do should be to modify Tralics so that this style sheet can be made more robust.

```
<xsl:template match="figure/p">
   <tr><xsl:apply-templates/></tr>
</xsl:template>
```

This copies the id attribute if present.[12]

```
<xsl:template name="id">
 <xsl:if test="./@id">
   <xsl:attribute name="id"><xsl:value-of select="./@id"/> </xsl:attribute>
 </xsl:if>
</xsl:template>
```

This interprets `<head>`. If the parent is `<list>`, we put the content of the element in the title attribute of the current element.[13] If the parent is `<figure>` or `<table>` we put the content in a `<caption>` element. Otherwise, we put it in a `<bodyTitle>` element. Note that Tralics replaces some empty titles by '`(Sans Titre)`'; in some cases they are replaced by 'Introduction'. You will see twice `Xsl` instead of `xsl`, in both these cases, the code contained a `<xsl:text></xsl:text>` that is not shown here (it seems useless to me).

```
<xsl:template match="head" mode="caption">
  <xsl:choose>
    <xsl:when test="parent::list" >
      <xsl:attribute name="title"> <xsl:apply-templates/> </xsl:attribute>
    </xsl:when>
    <xsl:when test="parent::figure | parent::table">
      <xsl:element name="caption"> <xsl:apply-templates/> </xsl:element>
    </xsl:when>
    <xsl:otherwise>
      <xsl:element name="bodyTitle">
        <xsl:choose>
          <xsl:when test=".='(Sans Titre)'">
            <xsl:choose>
              <xsl:when test="count(../../module)=1">
```

[12] Why not a simple copy-of ?

[13] This is strange; Tralics does not produce such a thing.

```
434                 <xsl:text>(Sans Titre)</xsl:text>
435               </xsl:when>
436               <xsl:otherwise> <Xsl:text>Introduction</xsl:text></xsl:otherwise>
437             </xsl:choose>
438           </xsl:when>
439           <xsl:otherwise> <Xsl:apply-templates/> </xsl:otherwise>
440         </xsl:choose>
441       </xsl:element>
442     </xsl:otherwise>
443   </xsl:choose>
444 </xsl:template>
```

We do nothing with <head>, because this element should be handled by the routines given above.

```
445 <xsl:template match="head"></xsl:template>
```

We leave the <LaTeX> element unchanged.

```
446 <xsl:template match="LaTeX"> <LaTeX/></xsl:template>
```

We leave the <TeX> element unchanged.

```
447 <xsl:template match="TeX"> <TeX/> </xsl:template>
```

This is the end of the file.

```
448 </xsl:transform>
```

5.2 Addings Ids

There is a style sheet that adds some Ids. It is really unclear why unreferenced elements should have an ID. This is the header of the file.

```
449 <xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
450   xmlns:m="http://www.w3.org/1998/Math/MathML"
451   exclude-result-prefixes="m">
```

The result is a XML file, conforming the Raweb DTD version 2.

```
452 <xsl:output method='xml' doctype-system='raweb2.dtd'
453      indent='yes' encoding='iso-8859-1'/>
```

We only use the first variable. This gives the name of the output file.

```
454 <xsl:variable name="LeProjet" select="/raweb/identification/@id"/>
455 <xsl:variable name="year"> unused </year>
```

The whole document is converted and printed in the file.

```
456 <xsl:template match="/">
457   <xsl:document href="{$LeProjet}.xml" method="xml"
458        doctype-system="raweb2.dtd" indent='yes' encoding='iso-8859-1'>
459     <xsl:apply-templates />
460   </xsl:document>
461 </xsl:template>
```

We copy all attributes, except id.

```
462 <xsl:template match="@id">
463 </xsl:template>
464
465 <xsl:template match="@*">
466   <xsl:copy />
467 </xsl:template>
```

If there is an id, we copy it, otherwise we invent one.

```
<xsl:attribute-set name="ID">
  <xsl:attribute name="id">
    <xsl:choose>
      <xsl:when test="./@id"> <xsl:value-of select="./@id" />  </xsl:when>
      <xsl:otherwise> <xsl:value-of select="generate-id()" />  </xsl:otherwise>
    </xsl:choose>
  </xsl:attribute>
</xsl:attribute-set>
```

Everything is copied, with an ID added.

```
<xsl:template match="*">
  <xsl:copy use-attribute-sets="ID">
    <xsl:apply-templates select="node()|@*" />
  </xsl:copy>
</xsl:template>

<xsl:template match="text()">
  <xsl:copy />
</xsl:template>
```

There is no need to add an ID to TeX and LaTeX.

```
<xsl:template match="LaTeX"> <LaTeX/> </xsl:template>
<xsl:template match="TeX">  <TeX/> </xsl:template>
```

Math formulas are left unchanged.

```
<xsl:template match="formula">
 <formula>
  <xsl:copy-of select="@*"/>
  <xsl:apply-templates mode="math"/>
 </formula>
</xsl:template>
```

This is standard.

```
<xsl:template match="*|@*|comment()|processing-instruction()|text()" mode="math">
 <xsl:copy>
  <xsl:apply-templates mode="math" select="*|@*|processing-instruction()|text()"/>
 </xsl:copy>
</xsl:template>
```

This is the end of the file.

```
</xsl:transform>
```

Chapter 6

Converting XML to HTML

The following set of style sheets is used to produce the HTML version of the Raweb. There are two possible views, topics, or not topics. Essentially the difference lies in the table of contents, and the order of pages. There are four style sheets: one for the case with topics, one for the case without topics, one for the bibliography, and one for everything else; we start with this one.

6.1 Common code for HTML conversion

This is the start of the style sheet.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0" xmlns:m="http://www.w3.org/1998/Math/MathML"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:html="http://www.w3.org/1999/xhtml"
  exclude-result-prefixes="m html xlink">
```

A very important point is the difference between a 'Project' and a 'Team'. Of the 160 teams that have written a RA in 2004, 110 were Projects at the start of the year. This code uses the isproject attribute of <identification>.

```
<xsl:variable name="LeTypeProjet">
  <xsl:if test='/raweb/identification/@isproject="false"'>Team</xsl:if>
  <xsl:if test='/raweb/identification/@isproject="true"'>Project</xsl:if>
</xsl:variable>
```

The global action is rather obvious: we call a routine for the title page, one for the presentation, one for every main subsection, one for the bibliography.

```
<xsl:template match="/raweb">
  <xsl:apply-templates select="identification" />
  <xsl:apply-templates select="identification/team" />
  <xsl:for-each select="presentation|fondements|domaine|logiciels|
          resultats|contrats|international|diffusion">
     <xsl:for-each select="subsection">
        <xsl:apply-templates select="." />
     </xsl:for-each>
  </xsl:for-each>
  <xsl:for-each select="biblio"> <xsl:apply-templates select="." /> </xsl:for-each>
</xsl:template>
```

6.1.1 Creating pages

Converting XML to HTML is rather easy, compared to conversion into Pdf. The non-trivial point concerns the layout of the page, navigation buttons, meta data, etc. One question is: should we put navigation buttons on the top and bottom of the page? If a page is large, it can be interesting to have a 'next' button near the end, this avoids the need to scroll to the top, if you want to continue reading. Starting in 1995, the Raweb was produced by `latex2html`, which can conditionnaly insert navigation buttons[1]; but this depends on the number of characters in the page, and not the effective size; as a consequence, the layout of the page seems to be random. The situation changed in 1999[2], the text is in a frame, and the navigation buttons are in an another frame, placed above the text; the buttons appear only in the first frame. In 2003[3] the situation changed again. There are two possible views. By default, there are no frames, but a button (the 'TOC' button) that creates two frames: the TOC on the left, the text on the right. The text has navigation buttons on the top and the bottom (but there are fewer buttons on the bottom). The same idea is used in 2004 and explained in this document. See figure 6.2.

Here is a helper for producing an `<img>`. It takes two arguments `$alt` and `$src`. Icons are in a shared directory.

```
519 <xsl:template name="icon.image">
520   <xsl:param name="alt" />
521   <xsl:param name="src" />
522   <img align="bottom" border="0" alt="{$alt}" src="../icons/{$src}.gif"></img>
523 </xsl:template>
```

The next command takes three arguments, `$nom`, `$position`, and `$accesskey`. If `$nom` is empty, the result is a simple image (for instance named `next_motif_gr.gif`, where 'gr' stands for 'grey' instead of black and white). Otherwise, it is the name of a HTML file, and the result is an anchor `<a>`, whose href attribute is this file name. The image is different (for instance named `next_motif.gif`, the images are designed so that it is obvious that they have the same purpose, one of them being active, the other inactive), but the alt field is the same; the accesskey[4] attribute is set only in this case. In any case, there is some white space after the button. Note: in certain case, a anchor has a target attribute. This can be `_alt` (this is a name not recognised by the standard; the effect is that the browser opens a new window); this can be `_top` or `_parent` (these names are defined by the standard); it can also be `mainraweb2004` (this is needed for links from the toc to the main frame). The buttons created by this procedure are in the main frame and point to the main frame; they have no target attribute.

```
524 <xsl:template name="make.icon">
525   <xsl:param name="nom" />
526   <xsl:param name="position" />
527   <xsl:param name="accesskey" />
528   <xsl:choose>
529    <xsl:when test="$nom=''">
530      <xsl:call-template name="icon.image">
531         <xsl:with-param name="alt" select="$position" />
532         <xsl:with-param name="src" select="concat($position,'_motif_gr')" />
533      </xsl:call-template>
534    </xsl:when>
535    <xsl:otherwise>
536      <a href="{$nom}.html">
```

[1] See for instance http://www.inria.fr/rapportsactivite/RA96/algo/algo.html

[2] See for instance http://www.inria.fr/rapportsactivite/RA99/algo/algo.html.

[3] See for instance http://www.inria.fr/rapportsactivite/RA2003/algo2003/algo.html; this is the version without the TOC.

[4] If the acceskey is, say N, pressing down the alt key together with the letter N has teh same effect as clicking on the icon, at least with my browser.

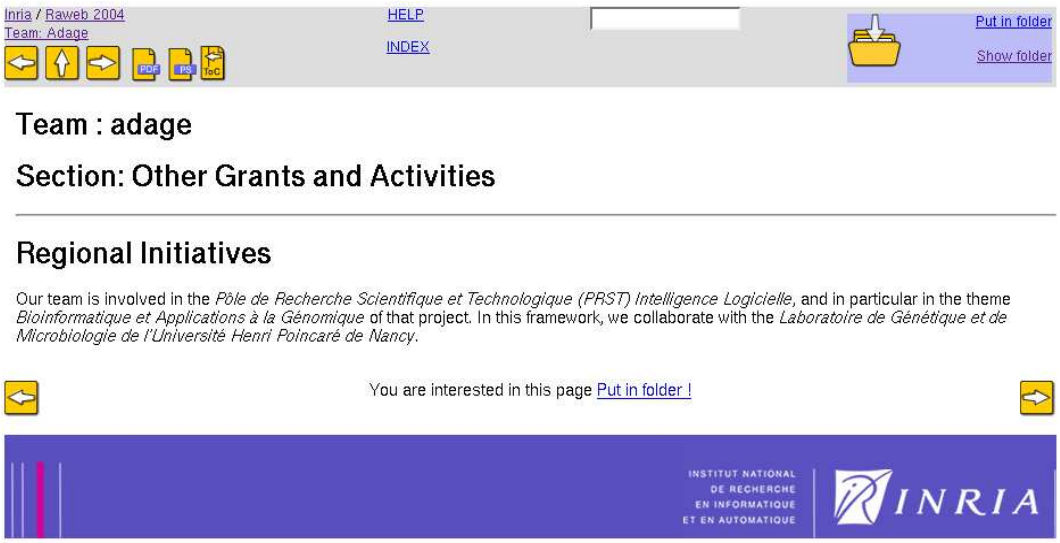

Figure 6.1: The layout of a sample page for a Team without topics.

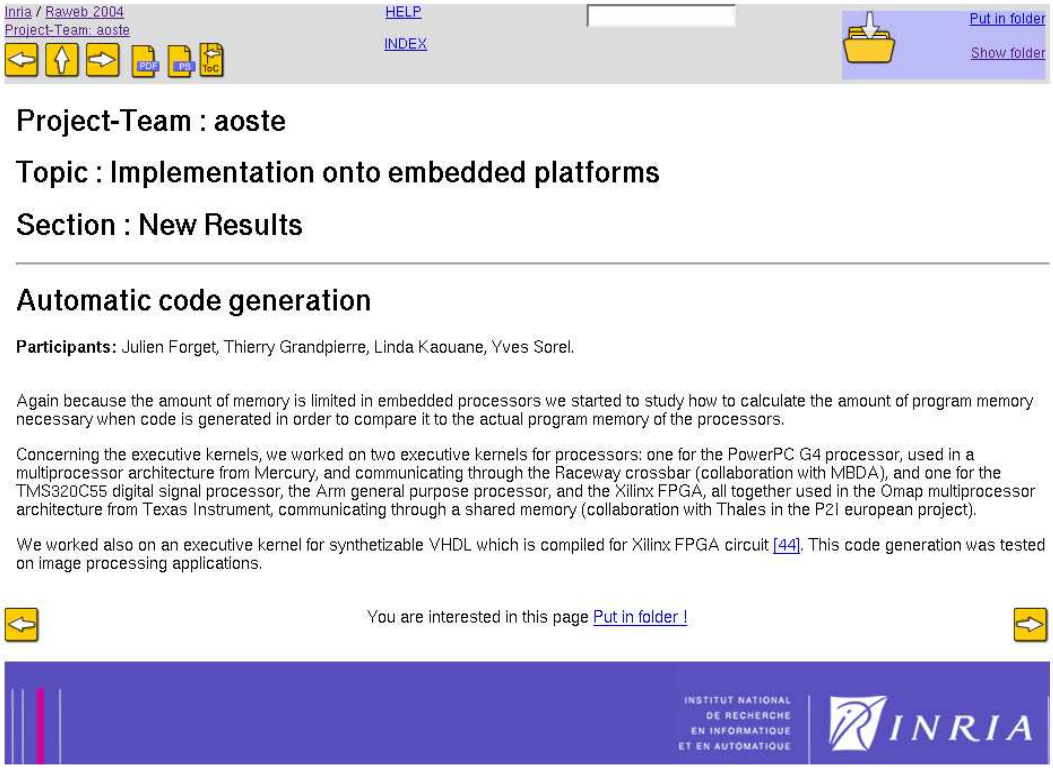

Figure 6.2: The layout of a sample page for a Project-Team with topics.

```
537         <xsl:attribute name="accesskey"><xsl:value-of select="$accesskey" />
538            </xsl:attribute>
539         <xsl:call-template name="icon.image">
540             <xsl:with-param name="alt" select="$position" />
541             <xsl:with-param name="src" select="concat($position,'_motif')" />
542         </xsl:call-template>
543        </a>
544      </xsl:otherwise>
545    </xsl:choose>
546    <xsl:text>
547
548    </xsl:text>
549 </xsl:template>
```

The next piece of code creates five buttons, and two other items, and puts them in a `<div>`. The first three are created by `make.icon`; this is something that takes three arguments; in order to make this document shorter, we have indicated only the value of the parameters, the names being `$nom`, `$position` and `$accesskey` in order. In the same fashion, `icon.image` is called with two parameters, named `$alt` and `$src`, we show only the value. The three quantities `$precedent`, `$suivant` and `$haut` are arguments to the procedure. They refer to the previous page, next page and top page, which may exist or not; if the page exists, the button is in an anchor, otherwise it is just an image. The following buttons are anchors to '../adage2004/adage.pdf' and '../adage2004/adage.ps.gz', assuming that `$year` contains 2004, and `$LeProjet` contains adage. The next item is an anchor, whose id is 'toclink', and whose href is something like 'adage_tf.html?../adage2004/uid4.html': after the question mark, there is the address of the current page, and before it is the name of the page with the frames. Then comes a javascript (it is defined in `lib.js`). The raweb.css file gives 'display:none' as property for the element identified by the `toclink` id. As a consequence, the button is invisible. However, if the name of the frame is 'mainraweb2004', the code on line 580 sets the 'display' style of elements with id 'toclink' to 'inline', so that the button is visible.

Assume that the button is visible, and you click on it. This will load the 'adage_tf.html' file; its content is explained later; all that you have to know is that the browser contructs a page with two frames, on the left the TOC, on the right is the current page (what follows the question mark); the name of this second frame is 'mainraweb2004', as a consequence the button is invisible.

The main Raweb location is http://www.inria.fr/rapportsactivite, this contains a subdirectory for each year, for instance RA2004. In this directory, we have some common files and directories (the css, the icons, etc.), and the teams, for instance adage2004. For a HTML page under adage2004, the next page can be foo.html or ../adage2004/foo.html, the style sheet is in ../raweb.css.

```
550 <xsl:template name="page.icons">
551  <div class="NavigationIcones">
552     <xsl:param name="precedent" />
553     <xsl:param name="suivant" />
554     <xsl:param name="haut" />
555     <xsl:call-template name="make.icon">("$precedent", "'previous'", "'P'")
556     </xsl:call-template>
557     <xsl:call-template name="make.icon">("$haut","'up'","'U'")
558     </xsl:call-template>
559     <xsl:call-template name="make.icon">("$suivant","'next'" ,"'N'")
560     </xsl:call-template>
561     <a href="../{$LeProjet}{$year}/{$LeProjet}.pdf">
562       <xsl:call-template name="icon.image"> ("'PDF'","'pdf_motif'")
563       </xsl:call-template>
564     </a>
565     <a href="../{$LeProjet}{$year}/{$LeProjet}.ps.gz">
566       <xsl:call-template name="icon.image"> ("'PS'","'ps_motif'")
```

```
      </xsl:call-template>
    </a>
    <a id="toclink">
       <xsl:attribute name="href">
         <xsl:value-of select="$LeProjet"/>_tf.html?../<xsl:value-of
             select="$LeProjet"/><xsl:value-of select="$year"/>/<xsl:value-of
             select="@id"/>.html</xsl:attribute>
       <img align="bottom" border="0" alt="Access to tdm"
          src="../icons/contents_motif.gif" />
    </a>
    <script type="text/javascript">
      <xsl:comment> var cible=this.location;
        if (self.name != "mainraweb04")
            changestyle('toclink','inline');</xsl:comment>
    </script>
 </div>
</xsl:template>
```

The bottom of the page contains only two buttons: previous and next. We have used the same conventions as above. The parameters `$precedent` and `$suivant` contain the locations of the previous and next pages. We insert another item, a javascript. These three items are each in a `<div>`, with an id attribute, the raweb.css file says that these should be flushed left or right, or centered. There is a `<br>` element before and after these three `<div>` elements. There is also an empty `<p>`.

```
<xsl:template name="pagedown.icons">
    <xsl:param name="precedent" />
    <xsl:param name="suivant" />
    <xsl:comment>FIN du corps du module</xsl:comment>
    <br/>
    <div id="tail_agauche">
      <xsl:call-template name="make.icon">("$precedent","'previous'","'P'")
      </xsl:call-template>
    </div>
    <div id="tail_adroite">
      <xsl:call-template name="make.icon"> ("$suivant","'next'","'N'")
      </xsl:call-template>
    </div>
    <div id="tail_aucentre"> <xsl:call-template name="classeurlink2" /> </div>
    <br /><p/>
</xsl:template>
```

The bottom of the page contains, between the previous and next button, a javascript `<div>` that reads: "You are interested in this page Put in folder !".

```
<xsl:template name="classeurlink2">
    <div class="folderLine">
      <script type="text/javascript" src="../classeur/classeurInOut.js" />
      <noscript><p>Using javascript allows access to folder</p></noscript>
    </div>
</xsl:template>
```

The top of the page contains, on the right, a `<div>` element (whose color is a kind of blue, the raweb.css file says it is BCBCF9), with an image and a pointer to the help page for the folder. There is also a `<script>`, that allows you to put the current page in the folder (same action as on the bottom of the page), or to view and manipulate the folder.

```
<xsl:template name="classeurlink1">
  <div class="folderButtons">
    <a id="folderIconRef" href="../classeur/aide.html">
```

```
609       <img id="folderIcon" src="../icons/folder.gif" alt="icon of folder" border="0"/></a>
610     <script type="text/javascript" src="../classeur/classeurInOutShow.js" />
611   </div>
612   <noscript> Using JavaScript allows access to folder </noscript>
613 </xsl:template>
```

The middle part of the head of the page consists in two buttons vertically aligned (because of the `<br>`) in a `<small>` element. If you click on them, you get the help, and the index. On the right of these, you will find the search form.

```
614 <xsl:template name="head-middle">
615  <div id="head_aucentre">
616   <xsl:call-template name="formRechercheExalead" />
617   <small>
618    <a href="../aide.html" target="aide" class="notcd" onclick="displayHelp(this)">HELP</a>
619    <br /> <br />
620    <a href="../index/index.html" target="_alt">INDEX</a>
621   </small>
622  </div>
623 </xsl:template>
```

This constructs the search `<form>`. We do not indicate here all the attributes, and the hidden fields.

```
624 <xsl:template name="formRechercheExalead">
625   <form ...>
626     Search in Activity Report, year <xsl:value-of select="$year" />:<br />
627     <input name="_q" size="20" maxlength="800" value=""/>
628   </form>
629 </xsl:template>
```

Another form. Why are two forms required? The search engine was AltaVista until July 2005, and Exalead after that. The version of the style sheet we present here is dated 2005/03/01. Apparently, this form is used, rather then the preceding one. The difference is tiny (of course, the result of the search engine is completely different, but this has nothing to do with the HTML input).

```
630 <xsl:template name="formRecherche">
631   <form id="recherche" ... >
632     <input NAME="q1" size="20" maxlength="800" VALUE=""/>
633   </form>
634 </xsl:template>
```

At the very bottom of the page, we have the Inria logo and a link to Inria's home page. The `<div>` and `<span>` elements have an id that is interpreted by the raweb.css style sheet.

```
635 <xsl:template name="bandeau_inria">
636   <div id="bandeau">
637     <div id="bandeau_filet"></div>
638     <a href="http://www.inria.fr"><span id="bandeau_logo" /></a>
639   </div>
640 </xsl:template>
```

This creates the top of the page. It takes four arguments, namely `$precedent`, `$haut`, `$suivant`, which are the names of the previous, up or next page, and `$couleur`. This last value is not a color, it is a symbolic name (it is 'premiere' or 'autre', French for 'first' and 'other') that will be added as class attribute to the main `<div>` element. This `<div>` element has three `<div>` children, that explain what should be put on the left, the middle, and the right of the page. We have already explained what is on the middle and the right. On the left, there are three lines, separated by `<br>`, using a `<small>` font. The first line contains something like "Inria / Raweb 2004", with two links, the second line contains "Team: Adage", with a link to the team's home page (note that 'Adage'

comes from <shortname>, and the link points to 'adage', that comes from $LeProjet). The last line contains the navigation buttons, created by page.icons (arguments are obvious, they are not indicated here, but replaced by a question mark).

```
<xsl:template name="bandeau-sup">
  <xsl:param name="couleur"/>
  <xsl:param name="precedent" />
  <xsl:param name="haut" />
  <xsl:param name="suivant" />
  <div id="toplign">
    <xsl:attribute name="class"><xsl:value-of select="$couleur" /></xsl:attribute>
    <div id="head_agauche">
     <small>
      <a href="http://www.inria.fr" target="_top">Inria</a> /
         <a href="../../index.html" target="_top">Raweb
            <xsl:value-of select="$year" /></a>
      <br />
      <a href="http://www.inria.fr/recherche/equipes/{$LeProjet}.en.html" target="_alt">
            <xsl:value-of select="$LeTypeProjet" />:
            <xsl:value-of select="/raweb/identification/shortname" /></a>
     </small>
     <br />
     <xsl:call-template name="page.icons"> (?,?,?)
     </xsl:call-template>
    </div>
    <xsl:call-template name="head-middle" />
    <div id="head_adroite"> <xsl:call-template name="classeurlink1" /> </div>
  </div>
</xsl:template>
```

This piece of code constructs the start of a page. In order to make things easier to understand, we split the code in two parts. In the first part, we construct three variables. One contains the authors: for each <person>, we take the <firstname> and <lastname>. The second variable contains the toplevel keywords, from <keyword>. The last variable contains the title, from <bodyTitle>.

```
<xsl:template name="page.head">
  <xsl:variable name="MTAUT">
     <xsl:for-each select=".//person">
        <xsl:value-of select="firstname" />
        <xsl:text> </xsl:text>
        <xsl:value-of select="lastname" />
        <xsl:text>/</xsl:text>
      </xsl:for-each>
  </xsl:variable>
  <xsl:variable name="MTMCL">
      <xsl:for-each select="./keyword">
        <xsl:apply-templates /> <xsl:text>/</xsl:text>
      </xsl:for-each>
  </xsl:variable>
  <xsl:variable name="MTDES"> <xsl:value-of select="./bodyTitle" /> </xsl:variable>
```

Action is trivial. The <title> will contain something like "Team-Adage", while dc.title contains the title of the document.

```
    <head>
      <title>
        <xsl:value-of select="$LeTypeProjet" />-<xsl:value-of
              select="/raweb/identification/shortname" />
      </title>
```

```
686       <link rel="Stylesheet" href="../raweb.css" type="text/css" />
687       <meta name="description" content="{$MTDES}" />
688       <meta name="dc.title" content="{$MTDES}" />
689       <meta name="dc.creator" content="{$MTAUT}" />
690       <meta name="dc.subject" content="{$MTMCL}" />
691       <meta name="dc.publisher" content="INRIA" />
692       <meta name="dc.date" content="(SCHEME=ISO8601) 2002-01" />
693       <meta name="dc.type" content="Report" />
694       <meta name="dc.language" content="(SCHEME=ISO639-1) en" />
695       <xsl:call-template name="classeurDecl"/>
696       <xsl:call-template name="interrogationDecl"/>
697     </head>
698   </xsl:template>
```

This declares the three javascripts.

```
699 <xsl:template name="interrogationDecl">
700     <script type="text/javascript" src="../interro.js" />
701 </xsl:template>
702 <xsl:template name="classeurDecl">
703     <script type="text/javascript"  src="../classeur/classeur.js" />
704     <script type="text/javascript"  src="../lib.js" />
705 </xsl:template>
```

We generate a HTML page for each module, one for the composition of the team (that comes before the first module), and one for the bibliography that comes last. The first page (the title page) will contain the full table of contents. The page with the composition of the team is the second page, it is constructed here, by applying a template to the `<team>` element. The links to the top and previous pages are identical, they point to the title page (its name comes from the variable `$LeProjet`), and the link to the next page is to the first `<subsection>` (the name of the page is the id attribute, with a '`.html`' extension). In order to reduce the size of this document, we do not show the parameters to `bandeau-sup` and `pagedown.icons`, the value is obvious, it is replaced by a question mark.

Assume that the id of the team is 'uid1', its name is 'adage' and the current year is 2004. Then the result is a HTML file, named adage2004/uid1.html, that contains a `<html>` element, that contains a `<body>` element[5]. We have explained above how page headers and footers are created, let's explain here how the body is constructed. First, we extract all `<moreinfo>` elements that come from this element or the `<identification>` part, they are output in a `<blockquote>` element. After that, we insert the title in a `<h1>` heading. This is followed by all the `<participants>` elements; each such element has a category attribute, that gives a subtitle, a `<h3>` element. We convert underscores back to spaces. Each `<person>` of these participants is inserted, with a `<br>` as separator.

```
706 <xsl:template match="identification/team">
707   <xsl:variable name="precedent" select="$LeProjet" />
708   <xsl:variable name="haut" select="$LeProjet" />
709   <xsl:variable name="suivant" select="/raweb/*/subsection[1]/@id" />
710   <xsl:document href="{$Directory}/{@id}.html"
711       method="xml" encoding="iso-8859-1"
712       indent="yes" doctype-public='-//W3C//DTD XHTML 1.0 Transitional//EN'
713       doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
714     <html>
715       <xsl:call-template name="page.head" />
716       <body>
717         <xsl:call-template name="bandeau-sup">
718           <xsl:with-param name="couleur" select="'autre'" /> (?,?,?)
```

[5] All HTML pages are created in this directory, or the other one with the topics.

```
719         </xsl:call-template>
720         <div id="main">
721           <xsl:for-each select="/raweb/identification/moreinfo">
722             <blockquote> <xsl:apply-templates /> </blockquote>
723           </xsl:for-each>
724           <xsl:for-each select="./moreinfo">
725               <blockquote> <xsl:apply-templates /> </blockquote>
726           </xsl:for-each>
727           <h1>Team</h1>
728           <xsl:for-each select="participants">
729             <h3> <xsl:value-of select="translate(@category, '_', ' ')" /> </h3>
730             <xsl:for-each select="person">
731                <xsl:apply-templates select="." />  <br />
732               </xsl:for-each>
733           </xsl:for-each>
734         </div>
735         <br />
736         <xsl:call-template name="pagedown.icons"> (?,?)
737         </xsl:call-template>
738         <xsl:call-template name="bandeau_inria"/>
739       </body>
740     </html>
741   </xsl:document>
742 </xsl:template>
```

In the case of a `<subsection>`, we distinguish between sections of level one or below (originally called `<module>` or `<div?>`). Let's first consider the case where the subsection is not in another subsection. In this case, we generate a page, like above (arguments of `bandeau-sup` and `pagedown.icons` are not shown). The page body contains the following: it starts with a title (for instance "Team : adage" in a `<h2>`, followed by the title of the section (it could be "Section: Overall Objectives" in a `<h2>`), followed by the keywords, followed by a horizontal rule, a `<hr>`. This is followed by the title (value of `<bodyTitle>`) in a `<h2>`, unless this is a fake title. Then comes the content of the subsection[6].

```
743 <xsl:template match="subsection">
744  <xsl:variable name="haut" select="$LeProjet" />
745  <xsl:variable name="precedent"><xsl:call-template name="precedent" /> </xsl:variable>
746  <xsl:variable name="suivant"><xsl:call-template name="suivant" /></xsl:variable>
747   <xsl:choose>
748    <xsl:when test="not(parent::subsection)">
749      <xsl:document href="{$Directory}/{./@id}.html" method="xml"
750         indent="yes" encoding="iso-8859-1"
751         doctype-public='-//W3C//DTD XHTML 1.0 Transitional//EN'
752         doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
753        <html>
754          <xsl:call-template name="page.head" />
755          <body>
756            <xsl:call-template name="bandeau-sup">(?,?,?)
757              <xsl:with-param name="couleur" select="'autre'" />
758            </xsl:call-template>
759            <div id="main">
760              <xsl:call-template name="use-id" />
761              <h2>
762                   <xsl:value-of select="$LeTypeProjet" />
```

[6] This is a strange use of headings. The `<h1>` is used only for the word 'Team' (line 727), the word 'Bibliography' (line 811), the full name of the team on the title page (line 856), and module titles, but only in the case of topics, lines 952 and 1890)

```
                  <xsl:text> : </xsl:text>
                  <xsl:value-of select="$LeProjet" />
              </h2>
              <xsl:call-template name="section_title" />
              <hr />
              <xsl:call-template name="jg.keywords" />
              <xsl:if test="bodyTitle!='(Sans Titre)'">
                 <xsl:apply-templates select="bodyTitle" mode="titre2"/>
              </xsl:if>
              <xsl:apply-templates />
            </div>
            <xsl:call-template name="pagedown.icons"> (?,?)
            </xsl:call-template>
            <xsl:call-template name="bandeau_inria"/>
          </body>
        </html>
      </xsl:document>
    </xsl:when>
```

In the case where `<subsection>` is in a subsection, we output its title (value of `<bodyTitle>`) using `<h3>` or `<h4>`, depending on whether the parent is in a subsection, then the keywords, then the content.

```
    <xsl:otherwise>
      <xsl:call-template name="use-id" />
      <xsl:choose>
        <xsl:when test="(../parent::subsection)">
          <xsl:apply-templates select="bodyTitle" mode="titre4"/>
        </xsl:when>
        <xsl:otherwise>
           <xsl:apply-templates select="bodyTitle" mode="titre3"/>
        </xsl:otherwise>
      </xsl:choose>
      <xsl:call-template name="keywords-list" />
      <xsl:apply-templates />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

This constructs the bibliography from `<biblio>`. We do not show the interesting part of the code, it is in the other file.

```
<xsl:template match="biblio">
  <xsl:variable name="precedent"
       select="preceding-sibling::*[1]/subsection[position()=last()]/@id"/>
  <xsl:variable name="haut" select="$LeProjet"/>
  <xsl:variable name="suivant" select="''"/>
  <xsl:document href="{$Directory}/bibliography.html"
        indent="yes" method="xml"
        doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
        doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
    <html>
      <xsl:call-template name="page.head"/>
      <body>
        <xsl:call-template name="bandeau-sup">(?,?,?)
           <xsl:with-param name="couleur" select="'autre'" />
        </xsl:call-template>
        <h1> Bibliography </h1>
        <xsl:call-template name="fill.the.bib"/>
```

```
813         <xsl:call-template name="pagedown.icons"> (?,?)
814         </xsl:call-template>
815         <xsl:call-template name="bandeau_inria"/>
816       </body>
817      </html>
818   </xsl:document>
819 </xsl:template>
```

This piece of code creates the title page, the TOC, and the frameset. In the case of the adage team, the files are adage.html, adage_tdm.html, and adage_tf.html in the adage2004 directory. Let's consider the title page first. It looks like a normal page, except that `bandeau_inria` (the banner with the Logo) is before the text.

```
820 <xsl:template match="identification">
821   <xsl:call-template name="creer.frameset" />
822   <xsl:variable name="precedent" select="''" />
823   <xsl:variable name="haut" select="$LeProjet" />
824   <xsl:variable name="suivant" select="team/@id" />
825   <xsl:document href="{$Directory}/{$LeProjet}.html" method="xml"
826       indent="yes" encoding="iso-8859-1"
827       doctype-public='-//W3C//DTD XHTML 1.0 Transitional//EN'
828       doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
829     <html>
830      <xsl:call-template name="page.head" />
831       <body>
832         <xsl:call-template name="bandeau-sup">
833            <xsl:with-param name="couleur" select="'premiere'" /> (?,?,?)
834         </xsl:call-template>
835         <xsl:call-template name="bandeau_inria"/>
836         <xsl:comment>DEBUT du corps du module</xsl:comment>
837         <xsl:call-template name="jg.titlepage"/>
838         <hr class="rose"/>
839         <xsl:call-template name="table.matieres" />
840         <xsl:call-template name="pagedown.icons"> (?,?)
841         </xsl:call-template>
842       </body>
843      </html>
844   </xsl:document>
```

This is the TOC. The page header is the same, but there are no navigation buttons. There is no page footer.

```
845   <xsl:document href="{$Directory}/{$LeProjet}_tdm.html"
846      method="xml" encoding="iso-8859-1" indent="yes"
847      doctype-public='-//W3C//DTD XHTML 1.0 Transitional//EN'
848      doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
849     <html>
850        <xsl:call-template name="page.head" />
851        <body> <xsl:call-template name="tdm" /> </body>
852      </html>
853   </xsl:document>
854 </xsl:template>
```

The title page is divided in two parts, the first part contains the identification of the team, it is followed by the TOC. We start with a `<h1>` element that contains the full name of the team, from `<projectName>`. This is followed by the short name, from `<shortname>`. It is followed by "2004 research project activity reports", a link to the UR (Research Unit), a link to the team, to the PostScript and the Pdf version of the report.

```
<xsl:template name="jg.titlepage"/>
  <h1 class="center"> <xsl:value-of select="projectName" /> </h1>
  <div class="entete">
    <div class="bigspace"><h2><xsl:value-of select="shortname" /></h2></div>
    <xsl:value-of select="$year" /> research project activity reports
    <xsl:call-template name="UR"/>
    <div class="bigspace">
      Theme:
        <a href="http://www.inria.fr/recherche/equipes/listes/
                                  theme_{substring(theme,1,3)}.en.html"
             TARGET="_alt">
             <xsl:value-of select="theme" />
        </a>
    </div>
    <a href="http://www.inria.fr/recherche/equipes/{$LeProjet}.en.html"target="_alt">
      <xsl:choose>
        <xsl:when test='/raweb/identification/@isproject="true"'>
              Presentation of the project</xsl:when>
        <xsl:otherwise>Presentation of the team</xsl:otherwise>
      </xsl:choose>
    </a> - Activity report in
         <a href="{$LeProjet}.ps.gz">PostScript</a> or
         <a href="{$LeProjet}.pdf">PDF</a> format
  </div>
</xsl:template>
```

Translation of <UR>. We look at the name attribute. The code is easy, perhaps a bit longish (we do not show the complete code here). We test the value against Rocquencourt, Rennes, Sophia, Lorraine, RhoneAlpes and Futurs.

```
<xsl:template name="UR">
 <xsl:variable name="orga">http://www.inria.fr/inria/organigramme/fiche</xsl:variable>
 <div class="bigspace">
  <xsl:for-each select="UR">
    <i>
    <xsl:choose>
     <xsl:when test="@name='Rocquencourt'">
      <a href="{$orga}_ur-rocq.en.html" target="_alt"><xsl:value-of select="@name"/></a>
     </xsl:when>
      <!-- other cases are similar, not shown -->
     <xsl:otherwise>
       <a href="http://www.inria.fr/inria/organigramme" target="_alt">INRIA</a>
     </xsl:otherwise>
    </xsl:choose>
   </i>
   <xsl:if test="position()!=last()"> - </xsl:if>
  </xsl:for-each>
 </div>
</xsl:template>
```

This contains the HTML page with the frameset. An interesting point is that this contains the table of contents, for the case where frames are refused. Note that this is an HTML document, not an XML one (the method attribute is differs); more important, all other HTML files have XHTML1.0 as doctype, this one has HTML4.01. The document contains two frames; one of them points to an HTML page whose name is dynamically constructed. The idea is the following. Normally, the `location.search` variable is empty and the result of line 917 is the name of the team; however, in the case of an URL like line 571, that contains a question mark, the variable is not empty, what follows the question mark is used.

```
<xsl:template name="creer.frameset">
 <xsl:document href="{$Directory}/{./@id}_tf.html" method="html" encoding="iso-8859-1"
  doctype-public="-//W3C//DTD HTML 4.01 Transitional//EN"
  doctype-system="http://www.w3.org/TR/html4/loose.dtd">
  <html>
   <head>
     <title>
       <xsl:value-of select="$LeTypeProjet" />:
       <xsl:value-of select="$LeProjet" />
     </title>
     <meta HTTP-EQUIV="Content-Type" content="text/html; charset=iso-8859-1" />
     <meta name="Robots" content="noindex" />
     <meta name="Generator" content="LaTeX2HTML v2K.1beta" />
     <meta HTTP-EQUIV="Content-Style-Type" CONTENT="text/css" />
     <link rel="Stylesheet" href="../raweb.css" type="text/css" />
     <style type="text/css"> div.tdmdiv { width:100% } </style>
     <script type="text/javascript">
        contenuSRC = (location.search.substring(1))
          ?location.search.substring(1) : '<xsl:value-of select="$LeProjet" />.html';
        contenuSRC = unescape(contenuSRC);
        var writeFrame = '';
        writeFrame += '&lt;frameset COLS="235,*"&gt;';
        writeFrame += '&lt;frame src="<xsl:value-of select='$LeProjet' />_tdm.html"&gt;';
        writeFrame += '&lt;frame src="' + contenuSRC +
                      '" name="mainraweb04" SCROLLING="auto" &gt;';
        writeFrame += '&lt;\/frameset&gt;';
        document.write(writeFrame);
     </script>
   </head>
   <noscript>
     <h1 class="warning">Using javascript is better to read this Activity Report</h1>
     <xsl:call-template name="tdm" />
   </noscript>
  </html>
 </xsl:document>
</xsl:template>
```

6.1.2 Titles, keywords, persons

This outputs the current section title, the <bodyTitle> of the parent, using <h2> as heading.

```
<xsl:template name="section_title">
  <h2>Section: <xsl:value-of select="../bodyTitle" /></h2>
</xsl:template>
```

This outputs the current title, found in <bodyTitle>, using <h2>, <h3>, or <h4> as heading.

```
<xsl:template match="bodyTitle" mode="titre2">
  <h2> <xsl:apply-templates/></h2>
</xsl:template>
<xsl:template match="bodyTitle" mode="titre3">
  <h3> <xsl:apply-templates/></h3>
</xsl:template>
<xsl:template match="bodyTitle" mode="titre4">
  <h4><xsl:apply-templates/> </h4>
</xsl:template>
```

In the case of topics, the layout is a bit different, and titles are handled by the routines that follow. At level one, the result is a <h1>. If the title is missing, it will be replaced by the title of

the section (presentation, fondements, etc.). The result is empty if the parent has no topic. Note: Tralics refuses to create a section with an empty title. In some cases, for instance if this is the first module in a sequence of more than one modules, the title will be 'Introduction'. Otherwise, it will be '(Sans Titre)' and the style sheet is assumed to do something in this case.

```
<xsl:template match="(presentation|fondements|domaine|logiciels
  |resultats|contrats|international|diffusion)/subsection/bodyTitle"
    priority="1">
    <xsl:call-template name="use-id" />
    <xsl:if test="../subsection/@topic">
      <h1>
        <xsl:choose>
          <xsl:when test=".='(Sans Titre)'">
            <xsl:value-of select="../../bodyTitle" />
          </xsl:when>
          <xsl:otherwise> <xsl:apply-templates /> </xsl:otherwise>
        </xsl:choose>
      </h1>
    </xsl:if>
  </xsl:template>
```

At level two, the result is a <h2>, without hacks.

```
<xsl:template match="(presentation|fondements|domaine|logiciels
  |resultats|contrats|international|diffusion)/subsection/subsection/bodyTitle"
    priority="1">
    <xsl:call-template name="use-id" />
    <h2> <xsl:apply-templates /> </h2>
</xsl:template>
```

At level three, the result is a <h3>.

```
<xsl:template match="(presentation|fondements|domaine|logiciels|resultats|
   contrats|international|diffusion)/subsection/subsection/subsection/bodyTitle"
    priority="1">
    <xsl:call-template name="use-id" />
    <h3> <xsl:apply-templates /> </h3>
</xsl:template>
```

A title is always output by the routines shown above, hence the default action is empty.

```
<xsl:template match="bodyTitle"></xsl:template>
```

This is how we typeset a title in the TOC. We use a <li> element and a link to the page.

```
<xsl:template name="item_title_or_not_title">
 <li>
   <a href="{@id}.html" target="mainraweb04">
     <xsl:choose>
       <xsl:when test="bodyTitle='(Sans Titre)'">
          <xsl:value-of select="../bodyTitle" />
       </xsl:when>
       <xsl:when test="bodyTitle"> <xsl:value-of select="./bodyTitle" /> </xsl:when>
       <xsl:otherwise> <xsl:value-of select="../bodyTitle" /> </xsl:otherwise>
     </xsl:choose>
   </a>
 </li>
</xsl:template>
```

Section titles are typeset twice: in the text, and in the TOC. When we are in the TOC, ids are not added. This is not very important for the HTML, because the TOC is a separated file. For the Pdf version it is necessary; it could be necessary if we decided to add a local TOC to each page (à la minitoc, this was done, for instance in 1996).

```
<xsl:template match="bodyTitle" mode="tocsection">
  <xsl:apply-templates mode="section" />
</xsl:template>
```

This rule says that keywords are handled via `keywords-list`, except if we are in a toplevel subsection that has subsections, case where `keywords-list2` is used instead.

```
<xsl:template name="jg.keywords">
   <xsl:choose>
      <xsl:when test="not(./subsection)">
          <xsl:call-template name="keywords-list" />
      </xsl:when>
      <xsl:otherwise>
          <xsl:call-template name="keywords-list2" />
      </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

We modified a bit the code that follows: there was a `<p>`, moved inside the template 'listvir'[7]. In general, when a keyword is seen, we take all `<keyword>` elements from the subtree, and pass them to the template.

```
<xsl:template name="keywords-list">
  <xsl:if test=".//keyword">
     <xsl:call-template name="listvir">
       <xsl:with-param name='liste' select="(.//keyword)" />
     </xsl:call-template>
  </xsl:if>
</xsl:template>
```

However, in the case of a toplevel subsection (a module) with subsections, the action is different: we take only the `<keyword>` elements of this subsection.

```
<xsl:template name="keywords-list2">
 <xsl:if test="./keyword">
    <xsl:call-template name="listvir">
      <xsl:with-param name='liste' select="(./keyword)" />
    </xsl:call-template>
 </xsl:if>
</xsl:template>
```

This is the action when we have a list of keywords to convert. The test (is there a keyword on the tree?) should probably be replaced by a better one (is the list empty?). Keywords are in italics, the header is bold.

```
<xsl:template name="listvir">
   <xsl:param name="liste" />
   <p>
     <xsl:if test="./keyword"> <b>Keywords: </b> </xsl:if>
     <xsl:for-each select="$liste" >
       <i><xsl:apply-templates /></i>
       <xsl:call-template name="separateur.objet" />
     </xsl:for-each >
   </p>
</xsl:template>
```

The previous routines takes all keywords, so that the default action is empty.

```
<xsl:template match="keyword">  </xsl:template>
```

In the presentation part, a `<person>` is handled by this code. We output the `<firstname>`, a space, the `<lastname>`, and the `<moreinfo>`, provided that this is not empty. Brackets are added.

[7] This is a strange name. It has no obvious meaning

```
<xsl:template match="person">
 <xsl:value-of select="./firstname" />
 <xsl:text> </xsl:text>
 <xsl:value-of select="./lastname" />
 <xsl:if test="moreinfo">
   <xsl:if test="not(normalize-space(string(.)) =’’)">
      <xsl:text> [ </xsl:text>
      <xsl:apply-templates select="./moreinfo/node()" />
      <xsl:text> ] </xsl:text>
    </xsl:if>
  </xsl:if>
</xsl:template>
```

In the other sections, a `<person>` is similarly handled, but the code is a bit different.

```
<xsl:template name="xperson">
  <xsl:value-of select="./firstname"/>
  <xsl:text> </xsl:text>
  <xsl:value-of select="./lastname"/>
  <xsl:apply-templates select="moreinfo" mode="inpers"/>
</xsl:template>
```

If you look carefully, you can see that spaces around square brackets are different here and in the previous case.

```
<xsl:template match="moreinfo" mode ="inpers">
  <xsl:if test="not(normalize-space(string(.)) =’’)">
      <xsl:text> [</xsl:text><xsl:apply-templates/><xsl:text>]</xsl:text>
  </xsl:if>
</xsl:template>
```

This takes a list as argument, it outputs an s if there are more than one element in the list.

```
<xsl:template name="pluriel-p">
  <xsl:param name="liste" />
  <xsl:if test="count($liste)>1">s</xsl:if>
</xsl:template>
```

In a `<subsection>`, the translation of `<participants>` is formed of every `<person>`, with comma as separator and a double `<br>` at the end.

```
<xsl:template match="subsection//participants">
  <b>Participant<xsl:call-template name="pluriel-p"><xsl:with-param
         name="liste" select="person" /></xsl:call-template>: </b>
  <xsl:for-each select="person">
    <xsl:call-template name="xperson" />
    <xsl:call-template name="separateur.objet" />
  </xsl:for-each>
  <br /><br />
</xsl:template>
```

6.1.3 Other elements

Let’s consider some trivial commands. We leave `<i>` unchanged.

```
<xsl:template match="i">
  <i><xsl:apply-templates /></i>
</xsl:template>
```

Ditto for `<b>`.

```
<xsl:template match="b">
 <b> <xsl:apply-templates /> </b>
</xsl:template>
```

Ditto for <tt>.

```
<xsl:template match="tt">
  <tt><xsl:apply-templates /> </tt>
</xsl:template>
```

Ditto for <small>.

```
<xsl:template match="small">
  <small> <xsl:apply-templates /> </small>
</xsl:template>
```

Ditto for <big>.

```
<xsl:template match="big">
 <big> <xsl:apply-templates /> </big>
</xsl:template>
```

Ditto for <sup>.

```
<xsl:template match="sup">
  <sup> <xsl:apply-templates /> </sup>
</xsl:template>
```

Ditto for <sub>.

```
<xsl:template match="sub">
  <sub> <xsl:apply-templates /> </sub>
</xsl:template>
```

This code is the same as above. This is used for evaluation of a title in the TOC.

```
<xsl:template match="sup" mode="section">
  <sup> <xsl:apply-templates /> </sup>
</xsl:template>
<xsl:template match="sub" mode="section">
  <sub> <xsl:apply-templates /> </sub>
</xsl:template>
```

The translation of is a <i> in the case where the style attribute is 'UNDERLINE', is 'HIGHLIGHT' or is something else. We simplified a bit the code.

```
<xsl:template match="em">
    <i> <xsl:apply-templates /> </i>
</xsl:template>
```

The translation of is the same element. We copy the class attribute, but not the other ones (why?).

```
<xsl:template match="span">
 <span class="{@class}"> <xsl:apply-templates /> </span>
</xsl:template>
```

We could do better.

```
<xsl:template match="center">
  <xsl:apply-templates />
</xsl:template>
```

A <term> is converted into a <tt>.

```
<xsl:template match="term">
 <tt> <xsl:apply-templates /> </tt>
</xsl:template>
```

A <simplelist> is converted into a <ul>.

```
1101 <xsl:template match="simplelist">
1102   <ul> <xsl:apply-templates />  </ul>
1103 </xsl:template>
```

A <orderedlist> is converted into a <ol>.

```
1104 <xsl:template match="orderedlist">
1105   <ol> <xsl:apply-templates /> </ol>
1106 </xsl:template>
```

A <descriptionlist> is converted into a <dl>.

```
1107 <xsl:template match="descriptionlist">
1108   <dl> <xsl:apply-templates /> </dl>
1109 </xsl:template>
```

A <label> is converted into a <dt>.

```
1110 <xsl:template match="label">
1111  <dt> <xsl:apply-templates /> </dt>
1112 </xsl:template>
```

A <glosslist> is converted into a <dl>. It has a title, is put in a , preceded and followed by a horizontal rule, a <hr> element.

```
1113 <xsl:template match="glosslist">
1114   <hr />
1115   <div id="glossaire" style="margin-left: 2em;margin-right: 2em;">
1116     <dl title="Glossary"> <xsl:apply-templates /> </dl>
1117    </div>
1118   <hr />
1119 </xsl:template>
```

A <li> is converted into a <li> or <dd>, depending on the value of the parent.

```
1120 <xsl:template match="li">
1121  <xsl:choose>
1122    <xsl:when test="(parent::glosslist) or (parent::descriptionlist)">
1123     <dd> <xsl:apply-templates /> </dd>
1124    </xsl:when>
1125    <xsl:otherwise>
1126      <li> <xsl:apply-templates />  </li>
1127    </xsl:otherwise>
1128  </xsl:choose>
1129 </xsl:template>
```

The translation of <moreinfo> is a <blockquote>, unless it is in a <pers>, case where the code on lines 1033 or 1042 applies.

```
1130 <xsl:template match="moreinfo">
1131   <blockquote>  <xsl:apply-templates /> </blockquote>
1132 </xsl:template>
```

Action is empty here. Normally, there should be no <moreinfo> in <raweb>, it should be in <identification>.

```
1133 <xsl:template match="/raweb/moreinfo" priority="1"/>
```

The translation of <simplemath> is <i>. Such an element is generated by Tralics in the case of a trivial formula like x.

```
1134 <xsl:template match="simplemath">
1135   <i> <xsl:apply-templates /> </i>
1136 </xsl:template>
```

Math formulas are copied verbatim, with their context. We assume that either the browser understands MathML or that a postprocessor converts the math.

```
<xsl:template match="m:math">
  <xsl:copy>
      <xsl:apply-templates mode="math" />
  </xsl:copy>
</xsl:template>

<xsl:template mode="math" match="*|@*|text()">
 <xsl:copy>
    <xsl:apply-templates mode="math" select="*|@*|text()" />
 </xsl:copy>
</xsl:template>
```

This is for the case where the math appears in a title in the TOC.

```
<xsl:template match="m:math" mode="section">
  <m:math>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates mode="math" />
   </m:math>
</xsl:template>
```

Translation of `<formula>`. It depends on the type attribute. If this is not 'display', the result is a simple `<span>`, with attribute class='math'. If the type is 'display', the result is a `<div>`, with attribute class='mathdisplay', and align='center'. Moreover, if there is an id, an equation number is added. For this reason a `<table>` is created, with a single `<tr>` and two `<td>`, first the formula, centered, then its number, right aligned.

```
<xsl:template match="formula">
 <xsl:choose>
  <xsl:when test="@type = 'display' and @id">
   <div align="center" class="mathdisplay">
     <xsl:call-template name="use-id" />
     <table width='100%'>
      <tr valign='middle'>
       <td align='center'> <xsl:apply-templates /> </td>
       <td class="eqno" width="10" align="right">(<xsl:number
                                level="any" count="formula[@id]" />)</td>
      </tr>
     </table>
   </div>
  </xsl:when>
  <xsl:when test="@type = 'display'">
    <div align="center" class="mathdisplay"> <xsl:apply-templates /> </div>
  </xsl:when>
  <xsl:otherwise>
   <span class="math"> <xsl:apply-templates /> </span>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

The translation of `<footnote>Text</footnote>` is '(Text)', this is much more legible than a link to the text.

```
<xsl:template match='footnote'>
  <xsl:text>(</xsl:text><xsl:apply-templates /><xsl:text>)</xsl:text>
</xsl:template>
```

Because of the code above, a `<p>` in a footnote cannot be transformed into a paragraph.

```
<xsl:template match="footnote/p">
  <xsl:apply-templates />
</xsl:template>
```

The translation of <p> is a <blockquote> if the rend attribute is 'quoted'. Otherwise, it is a <p>. If rend is 'center' or 'centered', we add align='center'. If noindent is given[8], we add class='notaparagraph'.

```
<xsl:template match="p">
  <xsl:choose>
    <xsl:when test="@rend='quoted'">
        <blockquote> <xsl:apply-templates /> </blockquote>
    </xsl:when>
    <xsl:otherwise>
      <p>
        <xsl:choose>
          <xsl:when test="@rend='centered' or @rend='center'">
             <xsl:attribute name="align">center</xsl:attribute>
          </xsl:when>
          <xsl:when test="@noindent">
            <xsl:attribute name="class">notaparagraph</xsl:attribute>
          </xsl:when>
        </xsl:choose>
        <xsl:apply-templates />
      </p>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

This is a bit strange. Currently Tralics does not produce <anchor> elements.

```
<xsl:template name="use-id">
  <xsl:if test="@id"> <a name="{@id}" />  </xsl:if>
  <xsl:if test="anchor/@id"><a name="{anchor/@id}" /> </xsl:if>
</xsl:template>
```

Translation of <table>. We call some template. The result will be in a .

```
<xsl:template match="table">
 <div align='center'>
    <xsl:call-template name="use-id" />
    <xsl:call-template name='generate-table' />
 </div>
</xsl:template>
```

If the table is inline, there is no reference to it (we can omit the id), and we do not produce a .

```
<xsl:template match="table[@rend='inline']">
  <xsl:call-template name='generate-table' />
</xsl:template>
```

This is called when the table comes from an object. Note that the 'mode' is useless in the 'xsl:call-template'.

```
<xsl:template match="table" mode="object">
  <xsl:call-template name="use-id" />
  <xsl:call-template name='generate-table' mode="object" />
</xsl:template>
```

Translation of <table>. The result is a table, that contains the translation of the <caption>, followed by some <tr> elements. Each such elements is teh translation of a <tr>, it is formed

[8] We should test the value.

of the translation of all cells, the <td> and <th> sub-elements. The style attribute of each cell is copied, as well as the style of the row[9], in the case where the cell is empty, a height of three pixels is added. If an attribute rows or cols is present, it is transformed into rowspan or colspan. If a child of a cell is a <ressource>, the align attribute is set to center[10].

```
<xsl:template name="generate-table">
 <table>
  <xsl:apply-templates select="caption" />
  <xsl:for-each select="tr">
   <tr>
    <xsl:for-each select="td|th">
     <xsl:element name="{name()}">
      <xsl:attribute name="style">
        <xsl:if test="normalize-space(.) = ''">height:3px;</xsl:if>
        <xsl:value-of select="@style" />
        <xsl:value-of select="../@style" />
      </xsl:attribute>
      <xsl:if test="@cols">
        <xsl:attribute name="colspan"><xsl:value-of select="@cols" /></xsl:attribute>
      </xsl:if>
      <xsl:if test="@rows">
        <xsl:attribute name="rowspan"><xsl:value-of select="@rows" /></xsl:attribute>
      </xsl:if>
      <xsl:if test="ressource">
         <xsl:attribute name="align">center</xsl:attribute>
      </xsl:if>
      <xsl:apply-templates />
     </xsl:element>
    </xsl:for-each>
   </tr>
  </xsl:for-each>
 </table>
</xsl:template>
```

A style attribute is always copied.

```
<xsl:template match="@style">
  <xsl:attribute name="style"><xsl:value-of select="." /></xsl:attribute>
</xsl:template>
```

A align attribute is converted to a style attribute.[11]

```
<xsl:template match="@align">
 <xsl:attribute name="style">text-align:<xsl-value-of select="." /></xsl:attribute>
</xsl:template>
```

The translation of a <caption> in a table is a <caption> element. It contains the table number (it should be the same as the one given by calculateTableNumber).

```
<xsl:template match="table/caption" >
  <caption align="bottom">
    <strong>Table
      <xsl:number count="table[not(ancestor::object)]" level="any" />
        <xsl:text>. </xsl:text>
    </strong>
    <xsl:apply-templates />
```

[9] Question: what says the DTD about this style attribute?

[10] What if the initial XML document specifies a different horizontal alignment, something hidden in the style attribute?

[11] What happens if both style and align are given? who wins? In fact, the question has no sense, because Tralics never emits a 'align', so that this template shoud never be called.

```
1260   </caption>
1261 </xsl:template>
```

The translation of a <caption> in a table in a figure. The result is a simple <caption>. There should be no reference to it.

```
1262 <xsl:template match="object//table/caption" priority="1">
1263    <caption> <xsl:apply-templates /> </caption>
1264 </xsl:template>
```

Translation of <img>. We copy the attributes width, height, align, border, alt, and src. Note: the <img> element is not defined by the DTD. Thus, the author of the document should not create them; on the other hand, there is a post-processor that replaces some math formulas by images that match this usage.

```
1265 <xsl:template match="img">
1266  <img>
1267    <xsl:if test="@width">
1268      <xsl:attribute name="width"><xsl:value-of select="@width" /></xsl:attribute>
1269    </xsl:if>
1270    <xsl:if test="@height">
1271      <xsl:attribute name="height"><xsl:value-of select="@height" /></xsl:attribute>
1272    </xsl:if>
1273    <xsl:attribute name="align"><xsl:value-of select="@align" /></xsl:attribute>
1274    <xsl:attribute name="border"><xsl:value-of select="@border" /></xsl:attribute>
1275    <xsl:attribute name="alt"><xsl:value-of select="@alt" /></xsl:attribute>
1276    <xsl:attribute name="src"><xsl:value-of select="@src" /></xsl:attribute>
1277   </img>
1278 </xsl:template>
```

Translation of <ressource>. The attribute media is 'WEB' by default. No rule applies otherwise.

```
1279 <xsl:template match="ressource[@media='WEB']">
1280   <xsl:call-template name='generate-graphics' />
1281   <xsl:apply-templates select="caption" />
1282 </xsl:template>
```

There is something strange here. If you specify both width and height, only the last attribute will be put in style. Perhaps, something like the code on lines 1226-1230 should be used instead. Note that the value of the aux attribute is not the name of the original image, but its conversion to gif or png.

```
1283 <xsl:template name="generate-graphics">
1284   <img>
1285     <xsl:if test="@width">
1286      <xsl:attribute name="style">width:<xsl:value-of select="@width" /></xsl:attribute>
1287       </xsl:if>
1288    <xsl:if test="@height">
1289       <xsl:attribute name="style">height:<xsl:value-of select="@height" /></xsl:attribute>
1290    </xsl:if>
1291    <xsl:attribute name='alt'><xsl:value-of select="@xlink:href" /></xsl:attribute>
1292    <xsl:attribute name='src'>../<xsl:value-of select="$LeProjet"/><xsl:value-of
1293             select="$year" />/<xsl:value-of select="@aux" /></xsl:attribute> </img>
1294 </xsl:template>
```

Translation of <object>. The result is a , that contains a <table>. Each table in the object is converted into a single <td> in a <tr>.

```
1295 <xsl:template match="object">
1296  <div align='center' style='margin-top:10px'>
1297     <a name="{./@id}"></a>
```

```
    <table title="{@title}" class="objectContainer">
       <xsl:call-template name="objectCaption" />
       <xsl:for-each select="table">
          <tr align="center">
            <td> <xsl:apply-templates select="." mode="object" /> </td>
          </tr>
       </xsl:for-each>
    </table>
  </div>
</xsl:template>
```

Translation of a <caption> of a <object>. It is like the caption of a table, but with name 'Figure' instead of 'Table'.

```
<xsl:template name="objectCaption">
  <caption align='bottom'>
      <strong>Figure
        <xsl:call-template name="calculateObjectNumber" />
        <xsl:text>. </xsl:text></strong>
      <xsl:apply-templates select="caption" />
  </caption>
</xsl:template>
```

If the caption is in a ressource in a cell, we output a <br>, and the value of the caption.

```
<xsl:template match="td/ressource/caption" priority="1">
  <br />
  <xsl:apply-templates />
</xsl:template>

```

If the caption is in a ressource, we output the value of the caption.

```
<xsl:template match="ressource/caption">
  <xsl:apply-templates />
</xsl:template>

```

If the caption matches none of the rules above, we output the value of the caption.

```
<xsl:template match="caption">
  <xsl:apply-templates />
</xsl:template>
```

These two elements exist so that we can transform them into TEX or LATEX in the Pdf result. In the HTML, it is better to use letters only.

```
<xsl:template match="LaTeX">LaTeX</xsl:template>
<xsl:template match="TeX">TeX</xsl:template>
```

This is a helper function. It adds a comma if the element is not the last.

```
<xsl:template name="separateurED.objet">
  <xsl:if test="position() &lt; last()">, </xsl:if>
</xsl:template>
```

This is a helper function. It adds a comma if the element is not the last, a dot otherwise.

```
<xsl:template name="separateur.objet">
  <xsl:choose>
    <xsl:when test="position() &lt; last()">, </xsl:when>
    <xsl:otherwise>.</xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

6.1.4 References

The code that follows (lines 1386 to 1393) contains lines of the form:

```
1339   <xsl:when test="count($curelement|key('bibliotypes', 'e'))=
1340                count(key('bibliotypes', 'e'))">
1341     <xsl:call-template name="positionInBib">
1342         <xsl:with-param name="str" select="'e'" />
1343         <xsl:with-param name="current" select="$curelement" />
1344         <xsl:with-param name="countPrevious" select="$d" />
1345     </xsl:call-template>
1346   </xsl:when>
```

We shall simplify this, writing `<positioninBib 'e' "$d"/>` instead of these eight lines. Here the test compares two values, `count(A|B)` and `count(B)`. The test is true if A is in B (A is a node, B is a node-set, `A|B` is the union of B and the set that contains only A). Such a construct is given as an example in [4] under the entry `<xsl:key>`. It happens that bibliography entries are sorted, by type, a type is a letter between d and k, and, for each type, by author. Consider the n-th item, let's call it X; question: what is n? Assume that the entry is of type m, and there are p entries of type less than m; if this entry is the q-th of type m, then $n = p + q$. All entries of type m can be found using `key`, with argument 'bibliotypes' and m. The test in the routine above is then: is `$curelement` of type '`e`'? If so, we call the template with three arguments: `$str` that holds the type, `$current` the entry to test, and `$countPrevious` the quantity p. The variable `$d` holds the number of entries of type 'd', it is computed on line 2138. Thus, we obtain p by adding some of `$d`, `$e`, `$f`, `$g`, `$h`, `$i`, and `$j` (the quantity `$k` is not used). Computing q is not trivial: we have a function that computes the index of the current node Y in a list, it is the `position()` function. We want the position of X, we obtain it by concatenation of all $f(Y)$, where $f(Y)$ is the position of Y if Y is X, empty otherwise. The nodes X and Y are the same if they have the same unique id, not if they have the same text. Normally, the number created here is the same as the one computed on line 2166 (the number associated to the entry), it is created for each reference to the bibliography.

```
1347 <xsl:template name="positionInBib">
1348   <xsl:param name="str" />
1349   <xsl:param name="countPrevious" />
1350   <xsl:param name="current" />
1351   <xsl:for-each select="key('bibliotypes',$str)">
1352     <xsl:sort select="descendant::author[1]/persName[1]/surname/text()" />
1353     <xsl:if test="generate-id(.)=generate-id($current)">
1354        <xsl:value-of select="$countPrevious+position()" />
1355     </xsl:if>
1356   </xsl:for-each>
1357 </xsl:template>
```

The translation of `<ref>` is an `<a>`. The non trivial point is to compute the value (the reference is much easier). If the link does not start with a `#`, it is an external link, this will be handled later. Otherwise some variables are used. First, `$curid` is the target id (the link without the `#`), and `$curelement` is the target. In Tralics, this can be a section, a math formula, an image, a table, a bibliographic entry, a footnote, an item in a list. In the Raweb, other elements might be referenced. We consider three variables: `$cursubsection`, `$curbiblio` and `$curidentification`, that contains ancestors of the target. These variables hold nodesets; their intersection is empty.

```
1358 <xsl:template match="ref">
1359   <xsl:choose>
1360     <xsl:when test="starts-with(@xlink:href, '#')">
1361       <xsl:variable name="curid"
1362           select="substring-after(@xlink:href, '#')" />
1363         <xsl:variable name="curelement"
```

```
1364          select="/raweb//*[@id=$curid]" />
1365        <xsl:variable name="cursubsection"
1366          select="$curelement/ancestor-or-self::subsection" />
1367        <xsl:variable name="curbiblio"
1368          select="$curelement/ancestor-or-self::biblio" />
1369        <xsl:variable name="curidentification"
1370          select="$curelement/ancestor-or-self::identification" />
```

Case one. `$curidentification` is not empty. This assumes that only one target exists in the page. A safer solution would be to replace `$curid` by the id of the `<team>` element. Case two, the target is in a subsection. We consider the first subsection (in document order), its id gives the name of the HTML page, it suffices to add the local part. The content of the link is computed elsewhere. In the third case, the target is in the bibliography. Otherwise, let's hope that the target is a section, we use the first subsection of it.

```
1371    <xsl:choose>
1372     <xsl:when test="$curidentification">
1373       <a href="./{$curid}.html">
1374         <xsl:apply-templates select="/raweb/identification" mode="xref" />
1375       </a>
1376     </xsl:when>
1377     <xsl:when test="$cursubsection">
1378        <a href="./{$cursubsection[1]/@id}.html{@xlink:href}">
1379            <xsl:apply-templates mode="xref" select="$curelement" />
1380        </a>
1381     </xsl:when>
1382     <xsl:when test="$curbiblio">
1383      <a href="bibliography.html{@xlink:href}">
1384        <xsl:text>[</xsl:text>
1385          <xsl:choose>
1386                <positioninBib 'd' "0"/>
1387                <positioninBib 'e' "$d"/>
1388                <positioninBib 'f' "$d+$e"/>
1389                <positioninBib 'g' "$d+$e+$f"/>
1390                <positioninBib 'h' "$d+$e+$f+$g"/>
1391                <positioninBib 'i' "$d+$e+$f+$g+$h"/>
1392                <positioninBib 'j' "$d+$e+$f+$g+$h+$i"/>
1393                <positioninBib 'k' "$d+$e+$f+$g+$h+$i+$j"/>
1394             </xsl:choose>
1395           <xsl:text>]</xsl:text>
1396      </a>
1397     </xsl:when>
1398     <xsl:otherwise>
1399      <a>
1400       <xsl:attribute name="href"><xsl:value-of
1401           select="$curelement/subsection[1]/@id" />.html</xsl:attribute>
1402       <xsl:apply-templates mode="xref" select="$curelement" />
1403      </a>
1404     </xsl:otherwise>
1405    </xsl:choose>
1406   </xsl:when>
```

In the case of an external reference, things are easier. The value of the `<ref>` element is assumed non-empty.[12]

```
1407   <xsl:otherwise>
1408    <a>
```

[12] This sets the target attribute of the anchor. This behaviour is sometimes annoying

```
1409        <xsl:attribute name="href"><xsl:value-of select="@xlink:href" /></xsl:attribute>
1410        <xsl:attribute name="target">_alt</xsl:attribute>
1411        <xsl:apply-templates />
1412      </a>
1413    </xsl:otherwise>
1414   </xsl:choose>
1415  </xsl:template>
```

For every element, like `<formula>`, that can be the target of a `<ref>`, we must compute its number. This is the value seen on the screen. In the case of a formula, this is easy, we just count all formulas with a number (i.e., with an id). Note: this behavior is different from what happens in the Pdf case. For this reason it would be safer to replace the code on line 1163 by a call to this template.[13]

```
1416  <xsl:template match="formula" mode="xref">
1417    <xsl:number level="any" count="formula[@id]" />
1418  </xsl:template>
```

In the case of a `<table>`, computation is non trivial. We use an intermediate command.

```
1419  <xsl:template match='table' mode="xref">
1420    <xsl:call-template name="calculateTableNumber" />
1421  </xsl:template>
```

This is how we count them: if the `<table>` is in an `<object>` it is not counted.

```
1422  <xsl:template name="calculateTableNumber">
1423     <xsl:number count="table[not(ancestor::object)]" level="any" />
1424  </xsl:template>
```

The same idea is used for `<object>` and `<ressource>`: we use a command.

```
1425  <xsl:template match="ressource" mode="xref">
1426    <xsl:call-template name="calculateRessourceNumber" />
1427  </xsl:template>
1428  <xsl:template match="object" mode="xref">
1429      <xsl:call-template name="calculateObjectNumber" />
1430  </xsl:template>
```

In order to understand this, remember that an `<object>` can contain a `<table>` that can contain some `<ressource>`. If an object contains more than one image, there is currently no way to refer to a specific image (Tralics allows you to reference the object, not the images). Hence, the number of a ressource is the number of its parent object.

```
1431  <xsl:template name="calculateObjectNumber">
1432    <xsl:number count="object" level="any" />
1433  </xsl:template>
1434
1435  <xsl:template name="calculateRessourceNumber">
1436    <xsl:number count="parent::object" level="any" />
1437  </xsl:template>
```

This returns the number of the section in which the element is. Note that 'fondements' are numbered 3, even if 'presentation' is missing. If you don't like this, then you should fill all sections[14].

```
1438  <xsl:template name="sec.num">
1439      <xsl:choose>
1440        <xsl:when test="ancestor-or-self::*[self::identification]"> 1</xsl:when>
1441        <xsl:when test="ancestor-or-self::*[self::presentation]">   2</xsl:when>
1442        <xsl:when test="ancestor-or-self::*[self::fondements]"> 3</xsl:when>
```

[13] If you look at the code line 487, you can see that conversion from one DTD to the other does not add ids to formulas. Hence, if Tralics adds no useless ids, you will see the same equation number in the Pdf and the HTML.

[14] Why is there a space before the number? The LaTeX equivalet would be wrong. It seems that my brower discards these spaces.

```
      <xsl:when test="ancestor-or-self::*[self::domaine]">  4</xsl:when>
      <xsl:when test="ancestor-or-self::*[self::logiciels]">  5</xsl:when>
      <xsl:when test="ancestor-or-self::*[self::resultats]">  6</xsl:when>
      <xsl:when test="ancestor-or-self::*[self::contrats]">   7</xsl:when>
      <xsl:when test="ancestor-or-self::*[self::international]"> 8</xsl:when>
      <xsl:when test="ancestor-or-self::*[self::diffusion]">  9</xsl:when>
      <xsl:when test="ancestor-or-self::*[self::biblio]">  10</xsl:when>
      <xsl:otherwise>*-</xsl:otherwise>
    </xsl:choose>
 </xsl:template>
```

The number associated to each section is given by the procedure above.

```
<xsl:template match="identification|presentation|fondements|domaine|
      logiciels|resultats|contrats|international|diffusion|biblio"
    mode="xref">
    <xsl:call-template name="sec.num" />
</xsl:template>
```

This computes the number of a subsection.

```
<xsl:template match="subsection" mode="xref">
  <xsl:call-template name="calculateNumbersubsection" />
</xsl:template>
```

We count the number of subsections, this is preceded by the section number. We removed here the variable `$numbersuffix`, replacing it by a simple `<xsl:text>` element.

```
<xsl:template name="calculateNumbersubsection">
   <xsl:call-template name="sec.num" />
   <xsl:text>.</xsl:text>
   <xsl:number level="multiple" grouping-separator="."
     from="raweb" format="1.1" count="subsection" />
</xsl:template>
```

In the case of a title or an anchor, we call a template (originally, there were two templates).

```
<xsl:template match="bodyTitle|anchor" mode="xref">
    <xsl:call-template name="calculateNumber" />
</xsl:template>
```

The number we compute in the generic case is the number of its subsection.

```
<xsl:template name="calculateNumber">
   <xsl:call-template name="sec.num" />
   <xsl:text>.</xsl:text>
   <xsl:number level="multiple" from="raweb" format="1.1" count="subsection" />
</xsl:template>
```

That's the end of the file.

```
</xsl:stylesheet>
```

6.2 The case without topics

This is the style sheet that converts an XML file into some HTML files. There are two variants; this is the easy one, because pages are inserted in the same order as in the original XML file. If the author of the document has used topics, then a second view is possible, as we shall see later.

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
  xmlns:m="http://www.w3.org/1998/Math/MathML"
  xmlns:xlink="http://www.w3.org/1999/xlink"
```

```
1480    xmlns:html="http://www.w3.org/1999/xhtml"
1481    exclude-result-prefixes="m html xlink" >
```

We include these two files. The content of the first file is explained above, the other one will be explained later. These files explain how to convert everything, the only exception concerns the table of contents: this is the essential difference between the two views, by topic or otherwise.

```
1482  <xsl:import href="common.xsl"/>
1483  <xsl:import href="biblio.xsl"/>
```

This seems standard, perhaps useless. Note that the doctype is XHTML elsewhere.

```
1484  <xsl:output method='html' encoding='iso-8859-1' doctype-public='-//W3C//DTD HTML 4.0//EN'/>
```

This had a certain utility last year. But these elements have now disappeared. The main reason why spaces are stripped is that, for a construct like '`<foo> <a/> <b/> </foo>`' it may be interesting to know that b is the last element in the list.

```
1485  <xsl:strip-space elements="cell bediteur bauteurs citation UR"/>
```

This seems useless.

```
1486  <xsl:param name="displaycd" />
```

This is the name of the team.

```
1487  <xsl:variable name="LeProjet" select="/raweb/identification/@id"/>
```

This the type of the team (Team or Project-Team).

```
1488  <xsl:variable name="LeTypeProjet">
1489   <xsl:choose>
1490     <xsl:when test='/raweb/identification/@isproject="true"'>Project-Team</xsl:when>
1491     <xsl:otherwise>Team</xsl:otherwise>
1492    </xsl:choose>
1493  </xsl:variable>
```

This is uselesss. It contains the same value as `$LeTypeProjet`, without accents (For a French version, it would contain Projet or Equipe, ASCII letters only).

```
1494  <xsl:variable name="LeTypeProjetSA">
1495  </xsl:variable>
```

Current year.

```
1496  <xsl:variable name="year">
1497   <xsl:choose>
1498     <xsl:when test='/raweb/@year'> <xsl:value-of select="/raweb/@year"/> </xsl:when>
1499     <xsl:otherwise>2004</xsl:otherwise>
1500   </xsl:choose>
1501  </xsl:variable>
```

The variable `$FTPDirectory` is useless.

```
1502  <xsl:variable name="FTPDirectory"    .../>
```

The `$Directory` variable is 'adage2004' here, and something else in the case of topics.

```
1503  <xsl:variable name="Directory" select="concat($LeProjet,$year)"/>
```

In order to make the document shorter, we have invented this template. It appears twice: here and in the other file. There is however a difference: if the case of a topic, the `<li>` is absent. This might be a bug. However, it is hard to notice. The main reason of the test here is that, in general, the presentation has only one module (subsection), with an empty title, that is replaced here by the name of the section. In any case, each module produces a `<li>` element, with a link to the main target.

```
1504  <xsl:template name="presentation_tdm_entry.JG">
1505    <ul>
1506     <xsl:for-each select="/raweb/presentation/subsection">
```

```
      <xsl:choose>
        <xsl:when test="bodyTitle='(Sans Titre)'">
          <li><a href="{@id}.html" target="mainraweb04">
              <xsl:value-of select="/raweb/presentation/bodyTitle" /></a></li>
        </xsl:when>
        <xsl:otherwise>
           <xsl:call-template name="item_title_or_not_title" />
        </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
  </ul>
</xsl:template>
```

This constructs a with the title and the content of the presentation.

```
<xsl:template name="presentation_tdm_entry">
  <div class="TdmEntry">
    <xsl:value-of select="/raweb/presentation/bodyTitle" />
    <xsl:call-template name="presentation_tdm_entry.JG" />
  </div>
</xsl:template>
```

This is some shared code: it prints the short name of the team, preceded by its type (Team or Project Team).

```
<xsl:template name="jg.insert-team-name">
 <div class="entete">
   <xsl:if test="/raweb/identification[@isproject]='true'">
      <xsl:text>Project </xsl:text></xsl:if>
   <xsl:text>Team </xsl:text>
   <xsl:value-of select="/raweb/identification/shortname" />
 </div>
</xsl:template>
```

This is the logo that appears in the TOC. There is no alt field in the case of topics... There is a link to Inria's home page.

```
<xsl:template name="jg.inria-logo">
 <div class="logo">
   <a href="http://www.inria.fr" target="mainraweb04">
     <img  border="0" src="../icons/logo-inria-.gif" alt="Inria" />
   </a>
 </div>
</xsl:template>
```

This is now the TOC. The structure is easy: we have, from top to bottom, the logo, the name of the team, the composition of the team, the presentation, the sections that might have a topic, and the bibliography.

```
<xsl:template name="tdm">
 <xsl:call-template name="jg.inria-logo"/>
 <div class="tdmdiv">
   <br />
   <xsl:call-template name="jg.insert-team-name"/>
   <div class="TdmEntry">
     <a href="{/raweb/identification/team/@id}.html" target="mainraweb04">Members</a></div>
   <xsl:call-template name="presentation_tdm_entry"/>
   <xsl:call-template name="table.tdm"/>
   <a href="bibliography.html" target="mainraweb04">Bibliography</a>
 </div>
</xsl:template>
```

There is no magic here. We simplified the code, by adding a variable, instead of looking at some strange location, in the case when the title of the module is empty.

```
<xsl:template name="table.tdm.entry">
  <xsl:variable name="varTitle" select="bodyTitle" />
  <div class="TdmEntry">
    <xsl:value-of select="./bodyTitle" />
    <ul>
      <xsl:for-each select="subsection">
       <xsl:choose>
         <xsl:when test="./bodyTitle='(Sans Titre)'">
           <li><a href="{@id}.html" target="mainraweb04">
               <xsl:value-of select="$varTitle" /> </a>
           </li>
         </xsl:when>
         <xsl:when test="./bodyTitle">
            <li> <a href="{@id}.html" target="mainraweb04">
                 <xsl:apply-templates mode="tocsection" select="bodyTitle" /</a>
            </li>
         </xsl:when>
         <xsl:otherwise>
           <li> <a href="{@id}.html" target="mainraweb04">
                    <xsl:value-of select="$varTitle" /></a>
           </li>
         </xsl:otherwise>
       </xsl:choose>
      </xsl:for-each>
    </ul>
  </div>
</xsl:template>
```

Let's consider the following sections: `<fondements>`, `<domaine>`, `<logiciels>`, `<resultats>`, `<contrats>`, `<international>`, and `<diffusion>`. We consider each module, and output it via the routine shown above. We start with the elements that have a topic, these are put in a `<div>`. This `<div>` is not created if there is no module with a topic. The CSS says that this element has a different color; it is preceded by a link (created by 'access.topic') that switches to the alternate view.

```
<xsl:template name="table.tdm">
  <xsl:if test="/raweb/child::*[self::fondements or
      self::domaine or self::logiciels or self::resultats or
      self::contrats or self::international or self::diffusion]//subsection[@topic]">
     <xsl:call-template name="acces.topic"/>
     <div class="topic">
      <xsl:for-each select="/raweb/child::*[self::fondements or
             self::domaine or self::logiciels or self::resultats or
             self::contrats or self::international or self::diffusion]">
          <xsl:if test=".//subsection[@topic]">
              <xsl:call-template name="table.tdm.entry" />
          </xsl:if>
      </xsl:for-each>
     </div>
  </xsl:if>
  <div class="non-topic">
    <xsl:for-each select="/raweb/child::*[self::fondements or
      self::domaine or self::logiciels or self::resultats or
        self::contrats or self::international or self::diffusion]">
          <xsl:if test="not(.//subsection[@topic])">
```

```
              <xsl:call-template name="table.tdm.entry" />
            </xsl:if>
    </xsl:for-each>
  </div>
</xsl:template>
```

This creates a `<div>` that contains two `<div>` elements, containing 'Default view' and 'View by topics'. These elements have different colors; the darker one contains a link. In the case of the Aoste Team, the link is to ../aoste2004_Topics/aoste_tf.html. Note that the target is `_parent`, so that the loading the aoste_tf.html file will remove this page and the sibling frame, replacing them by two pages (TOC and main page).

```
<xsl:template name="acces.topic">
  <br />
  <div id="bouton">
    <div id="clair">Default view</div>
    <div id="fonce">
      <a href="../{$LeProjet}2004_Topics/{$LeProjet}_tf.html"
            style="text-decoration:none" target="_parent">
         <div id="bouton_tdm_click">
           <font color="#FFFFFF">View by topics</font></div>
      </a>
    </div>
  </div>
  <br />
</xsl:template>
```

This piece of code is taken from the other style sheet. We put it here so that comparison is easy. There are two `<div>` in a `<div>`; the text is the same; the colors are not; the link points somewhere else.

```
<xsl:template name="acces.topic">
  <br />
  <div id="bouton">
    <div id="fonce_T">
      <a href="../{$LeProjet}{$year}/{$LeProjet}_tf.html"
         style="text-decoration:none" target="_parent">
         <div id="bouton_tdm_click">
             <font color="#FFFFFF">Default view</font></div>
      </a>
    </div>
    <div id="clair_T">View by topics</div>
  </div>
</xsl:template>
```

This creates the full table of contents, that is on the front page.

```
<xsl:template name="table.matieres">
 <ul class="tdm_frame">
  <li><a href="{/raweb/identification/team/@id}.html" target="mainraweb04"> Members </a>
  </li>
  <xsl:call-template name="table.section"/>
  <li> <a href="bibliography.html">Bibliography</a></li>
 </ul>
</xsl:template>
```

For each section, we insert its title and the list of its modules.

```
<xsl:template name="table.section">
  <xsl:for-each select="/raweb/child::*[self::presentation or self::fondements
      or self::domaine or self::logiciels or self::resultats or self::contrats or
```

```
      self::international or self::diffusion]">

      <xsl:variable name="varTitle" select="bodyTitle" />
      <li>
        <xsl:value-of select="bodyTitle" />
        <ul>
          <xsl:for-each select="subsection">
            <xsl:choose>
              <xsl:when test="./bodyTitle='(Sans Titre)'">
                <li>
                  <a href="{@id}.html" target="mainraweb04">
                    <xsl:value-of select="$varTitle" />
                  </a>
                </li>
              </xsl:when>
              <xsl:otherwise>
                <li>
                  <a href="{@id}.html" target="mainraweb04">
                    <xsl:apply-templates mode="tocsection"
                      select="bodyTitle" />
                  </a>
                </li>
              </xsl:otherwise>
            </xsl:choose>
          </xsl:for-each>
        </ul>
      </li>
    </xsl:for-each>
  </xsl:template>
```

Now comes the delicate part: what is the previous or next page? Finding the previous module is rather easy: if there is a previous module in the section, we chose it; if there is a previous section we chose the last module of it; otherwise it is the front page.

```
<xsl:template name="precedent">
  <xsl:choose>
    <xsl:when test="preceding-sibling::subsection">
      <xsl:value-of select="preceding-sibling::subsection[1]/@id" />
    </xsl:when>
    <xsl:when test="../preceding-sibling::*[1]/subsection">
      <xsl:value-of select="../preceding-sibling::*[1]/subsection[position()=last()]/@id"/>
    </xsl:when>
    <xsl:otherwise>
       <xsl:value-of select="/raweb/identification/team/@id" />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Finding the next module is easy too: if there is a next module in the section, we chose it; if there is a next section we chose the first module of it; otherwise it is the bibliography.

```
<xsl:template name="suivant">
  <xsl:choose>
    <xsl:when test="following-sibling::subsection">
      <xsl:value-of select="following-sibling::subsection[1]/@id" />
    </xsl:when>
    <xsl:when test="../following-sibling::*[1]/subsection">
      <xsl:value-of select="../following-sibling::*[1]/subsection[1]/@id" />
    </xsl:when>
```

```
1692      <xsl:otherwise>
1693        <xsl:text>bibliography</xsl:text>
1694      </xsl:otherwise>
1695    </xsl:choose>
1696  </xsl:template>
1697  <xsl:template match="subsection/bodyTitle">
1698  </xsl:template>
```

This is the end of the file.

```
1699  </xsl:stylesheet>
```

6.3 HTML with Topics

There is a second file that converts XML to HTML. It starts like this.

```
1700  <xsl:stylesheet
1701          xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
1702          xmlns:m="http://www.w3.org/1998/Math/MathML"
1703          xmlns:html="http://www.w3.org/1999/xhtml"
1704          exclude-result-prefixes="m html " >
```

It is followed by some lines, identical to lines 1481–1500 that we shall reproduce here. The only thing that changes is the following variable.

```
1705  <xsl:variable name="Directory" select="concat($LeProjet, $year,'_Topics')"/>
```

This constructs the toc on the main page. It consists in the following items: the composition of the team, the presentation (see below, the non-topic part thereof is selected), the text: "View by topics", all modules that have a topic, ordered by topics, surrounded by horizontal rules, followed by all modules that have no topic, including the bibliography. All links here have target=‘mainraweb04’.

```
1706  <xsl:template name="table.matieres">
1707    <div class="non-topic">
1708     <a href="{/raweb/identification/team/@id}.html" target="mainraweb04"> Members
1709     </a>
1710    </div>
1711    <xsl:call-template name="presentation_tdm_entry" />
1712    <hr class="topic_color" />
1713    <div align="right"><font color="green">View by topics<br/></font></div>
1714    <xsl:call-template name="table.topic">
1715          <xsl:with-param name="class">tdm_window</xsl:with-param>
1716    </xsl:call-template>
1717    <hr class="topic_color" />
1718    <ul class="tdm_window">
1719      <xsl:call-template name="table.section" />
1720      <li>
1721       <a href="{/raweb/biblio/@id}.html" target="mainraweb04">
1722             <xsl:value-of select="/raweb/biblio[1]/@titre"/>
1723       </a>
1724      </li>
1725    </ul>
1726  </xsl:template>
```

This is the TOC in the case of topics. There is a little difference with the code shown above: in fact, from the TOC frame you can toggle to the non-topic version.

```
1727  <xsl:template name="tdm">
1728   <xsl:call-template name="jg.inria-logo"/>
```

```
 <div class="tdmdiv">
   <BR/>
   <xsl:call-template name="jg.insert-team-name"/>
   <hr />
   <div class="non-topic">
     <a href="{/raweb/identification/team/@id}.html" target="mainraweb04"> Members  </a>
   </div>
   <xsl:call-template name="presentation_tdm_entry"/>
   <xsl:call-template name="acces.topic"/>
   <xsl:call-template name="table.topic">
      <xsl:with-param name="class">tdm_frame</xsl:with-param>
   </xsl:call-template>
   <xsl:call-template name="table.section"/>
   <a href="{/raweb/biblio/@id}.html" target="mainraweb04">
      <xsl:value-of select="/raweb/biblio[1]/@titre"/>
   </a>
 </div>
</xsl:template>
```

Note: it is assumed that either all modules in a section have a topic, or none. If this assertion fails, some module might be missing, or appear more than once.

This takes all modules from the eight main sections ('presentation' excepted) that have no topic, and creates a list item for them; it will contain the list of modules of the section.

```
<xsl:template name="table.section">
  <xsl:for-each select="/raweb/child::*[ self::fondements or self::domaine
     or self::logiciels or self::resultats or
     self::contrats or self::international or self::diffusion]">
    <xsl:if test="not(child::subsection/@topic)">
      <li>
        <xsl:value-of select="./bodyTitle" />
        <ul>
          <xsl:for-each select="subsection">
              <xsl:call-template name="item_title_or_not_title" />
          </xsl:for-each>
        </ul>
      </li>
    </xsl:if >
  </xsl:for-each>
</xsl:template>
```

This is the same code as in the non-topic case, but the attribute of the `<div>` is not the same, and the section title is missing.[15]

```
<xsl:template name="presentation_tdm_entry">
  <xsl:if test="not(/raweb/presentation/subsection/@topic)">
    <div class="non-topic">
      <xsl:call-template name=presentation_tdm_entry.JG"/>
    </div>
  </xsl:if>
</xsl:template>
```

For each `<topic>`, we output its name, in small caps, and for each section that has a module with this topic, we output the name of the section, and the relevant modules. The template takes a parameter, but it is not used.

[15] In the case where modules in the presentation section have topics, the algorithm for finding the previous module fails.

```
<xsl:template name="table.topic">
 <xsl:param name="class" />
 <div class="topic">
  <xsl:for-each select="/raweb/topic">
    <xsl:variable name="num-topic" select="@id" />
    <h3 class="smallcap"> <xsl:apply-templates /></h3>
   <xsl:for-each select="/raweb/child::*[self::presentation
            or self::fondements or self::domaine or
               self::logiciels or self::resultats or self::contrats or
               self::international or self::diffusion]">
    <xsl:if test="subsection[@topic=$num-topic]">
      <xsl:value-of select="bodyTitle" />
      <ul>
        <xsl:for-each select="subsection[@topic=$num-topic]">
          <xsl:call-template name="item_title_or_not_title" />
        </xsl:for-each>
      </ul>
    </xsl:if>
   </xsl:for-each>
  </xsl:for-each>
 </div>
 <br/>
</xsl:template>
```

This finds the previous module. There are two cases to consider: it depends on whether we are in the topic part, or non-topic part. We use two variables: `$var` is the current topic id and `$precedentTopic` is the previous topic (or is empty if there is no previous topic).

```
<xsl:template name="precedent">
  <xsl:variable name="var" select="./@topic" />
  <xsl:variable name="precedentTopic"
       select="/raweb/topic[@id=$var]/preceding-sibling::topic[1]" />
  <xsl:choose>
```

First assume that the module has a topic. The easy case is when there is a preceding sibling with the same topic. We chose the first one. Otherwise, we consider all modules that have as topic the previous topic; we select the last one; if this fails, we select the last module of the presentation section.

```
    <xsl:when test="./@topic">
      <xsl:choose>
        <xsl:when test="preceding::subsection[@topic=$var]">
          <xsl:value-of select="preceding::subsection[@topic=$var][1]/@id" />
        </xsl:when>
        <xsl:when test="/raweb/descendant::subsection[@topic=($precedentTopic/@id)]">
          <xsl:variable name="precedingTopicId" select="$precedentTopic/@id" />
          <xsl:variable name="lastSubsection"
             select="/raweb/descendant::subsection[@topic=($precedentTopic/@id)][last()]" />
          <xsl:value-of select="$lastSubsection/@id" />
        </xsl:when>
        <xsl:otherwise>
           <xsl:value-of select="/raweb/presentation/subsection[last()]/@id" />
        </xsl:otherwise>
      </xsl:choose>
    </xsl:when>
```

We consider now the case where the module has no topic. In the case where there is a preceding sibling in the same section without topic, we chose the first one. In the case where the section is the presentation, our module is located before the modules with topics, thus the preceding page is

that with the composition. If there is a preceding module without topic, we use it. Otherwise, if there are topics, and if there is a module with a topic, we chose the last module of the last topic. Otherwise, we chose the page with the team.

```
    <xsl:otherwise>
      <xsl:choose>
        <xsl:when test="parent::presentation and position()=1">
          <xsl:value-of select="/raweb/identification/team/@id" />
        </xsl:when>
        <xsl:when test="preceding-sibling::subsection[not(./@topic)]">

          <xsl:value-of select="preceding-sibling::subsection[not(./@topic)][1]/@id" />
        </xsl:when>
        <xsl:when test="../preceding-sibling::*[1]/subsection[not(./@topic)]">
          <xsl:value-of select="../preceding-sibling::*[1]/
                    subsection[not(./@topic)][last()]/@id" />
        </xsl:when>
        <xsl:when test="/raweb/topic and /raweb/descendant::subsection[@topic]">
          <xsl:variable name="lastTopic" select="//topic[last()]/@id" />
          <xsl:variable name="lastSubsection"
             select="/raweb/descendant::subsection[@topic=$lastTopic][last()]" />
          <xsl:value-of select="$lastSubsection/@id" />
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="/raweb/identification/team/@id" />
        </xsl:otherwise>
      </xsl:choose>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

This finds the next module. The algorithm is the same as above (to be precise: if X precedes Y, then Y follows X).

```
<xsl:template name="suivant">
  <xsl:variable name="var" select="./@topic" />
  <xsl:variable name="followingTopic"
     select="/raweb/topic[@id=$var]/following-sibling::topic[1]" />
  <xsl:choose>
```

There are two cases to consider. First assume that the module has a topic. The easy case is when there is a following sibling with a topic. We chose the first one. The next case is when there is a module with the following topic. We chose the first one. Then we consider the case where the current topic is the last one, and there is a module in one of the main sections (not presentation) that has no topic; the first one is selected. Otherwise, the next page is the bibliography.

```
    <xsl:when test="./@topic">
      <xsl:choose>
        <xsl:when test="following::subsection[@topic=$var]">
          <xsl:value-of select="following::subsection[@topic=$var][1]/@id" />
        </xsl:when>
        <xsl:when test="/raweb/descendant::subsection[@topic=$followingTopic/@id]">
          <xsl:value-of select="/raweb/descendant::subsection
                              [@topic=$followingTopic/@id][1]/@id" />
        </xsl:when>
        <xsl:when test="not($followingTopic) and
             /raweb/*[name()!='presentation']/subsection[not(@topic) or @topic='']">
          <xsl:value-of select="/raweb/*[name()!='presentation']/
                              subsection[not(@topic) or @topic=''][1]/@id" />
```

```
1858          </xsl:when>
1859          <xsl:otherwise>
1860            <xsl:text>bibliography</xsl:text>
1861          </xsl:otherwise>
1862        </xsl:choose>
1863      </xsl:when>
```

We consider here the case where the module has no topic. The easy case is when the section contains a following module without topic. Then comes the case where we are in the presentation section, it is the last module, and there is a module with a topic: in this case we chose the first module with the first topic. Otherwise we consider the case where there is a following module without topic, we chose the first one. Otherwise, we seclect the bibliography.

```
1864      <xsl:otherwise>
1865        <xsl:choose>
1866          <xsl:when test="following-sibling::subsection[not(@topic) or @topic='']">
1867            <xsl:value-of select="following-sibling::subsection[not(@topic)
1868                                                    or @topic=''][1]/@id" />
1869          </xsl:when>
1870          <xsl:when test="parent::presentation and (position()=last())
1871                 and /raweb/descendant::subsection[@topic and not(@topic='')]">
1872            <xsl:variable name="firstTopicId" select="/raweb/topic[1]/@id" />
1873            <xsl:value-of select="/raweb/descendant::subsection
1874                                          [@topic=$firstTopicId][1]/@id" />
1875          </xsl:when>
1876          <xsl:when test="../following-sibling::*[1]/subsection[not(@topic) or @topic='']">
1877
1878            <xsl:value-of select="../following-sibling::*[1]/subsection
1879                                        [not(@topic) or @topic=''][1]/@id" />
1880          </xsl:when>
1881          <xsl:otherwise>
1882            <xsl:text>bibliography</xsl:text>
1883          </xsl:otherwise>
1884        </xsl:choose>
1885      </xsl:otherwise>
1886    </xsl:choose>
1887  </xsl:template>
```

In the case where a module has a real title, we show the title of the section instead of the title of the module. Otherwise we do nothing.

```
1888  <xsl:template match="subsection[@topic]/bodyTitle">
1889   <xsl:if test=".!='(Sans Titre)'">
1890      <h1> <xsl:value-of select="../../bodyTitle" /> </h1>
1891    </xsl:if>
1892  </xsl:template>
```

In the case where the current element has a topic attribute, we output the value of the topic reference. Otherwise, we just output the section.

```
1893  <xsl:template name="section_title">
1894   <xsl:choose>
1895     <xsl:when test="./@topic">
1896        <xsl:variable name="VarTop" select="@topic" />
1897        <h2>Topic : <xsl:value-of select="/raweb/topic[@id=$VarTop]" /></h2>
1898     </xsl:when>
1899     <xsl:otherwise>
1900        <h2>Section : <xsl:value-of select="../bodyTitle" /></h2>
1901      </xsl:otherwise>
```

```
  </xsl:choose>
</xsl:template>
```

In all other cases, a body title disappears.

```
<xsl:template match="subsection/bodyTitle"></xsl:template>
```

This is the end of the file.

```
</xsl:stylesheet>
```

6.4 Converting the bibliography

This auxiliary file converts the bibliography from the old DTD to the new one.[16]

```
<xsl:transform
  xmlns:tei="http://www.tei-c.org/ns/1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">

<xsl:output method='xml' doctype-system='raweb3.dtd' indent='no'
   encoding='iso-8859-1'/>
```

We simplified the code, by assuming that the id is 'bibliography'. The transformation of <biblio> is an element of the same name. We consider all <citation> elements.

```
<xsl:template match="biblio">
  <xsl:element name="biblio">
    <xsl:attribute name="id">bibliography</xsl:attribute>
    <xsl:apply-templates select="citation"/>
  </xsl:element>
</xsl:template>
```

Translation of a <citation> element. The result is a <biblStruct>, depending on the type it can have one of two alternatives (on page 185, we have the template that converts the entry to HTML, and a comment that says that we check the validity; nothing is done here, we assume that the first or second if-test is true). The last three <note> elements will be used to sort the entries, or for debug.

```
<xsl:template match="citation">
  <xsl:element name="biblStruct">
    <xsl:attribute name="id"><xsl:value-of select="./@id"/></xsl:attribute>
    <xsl:if test="@type='article' or @type='inbook' or @type='incollection'
           or @type='inproceedings' or @type='conference'">
        <xsl:call-template name="citation.analy"/>
    </xsl:if>
    <xsl:if test="@type='book' or @type='booklet' or @type='proceedings' or
        @type='phdthesis'  or @type='techreport' or @type='unpublished' or
        @type='misc' or @type='masterthesis' or @type='mastersthesis' or
        @type='manual'">
      <xsl:call-template name="citation.mono"/>
    </xsl:if>
    <xsl:apply-templates select="bhowpublished"/>
    <xsl:element name="note">
      <xsl:attribute name="type">classification</xsl:attribute>
      <xsl:value-of select="./@type"/>
    </xsl:element>
    <xsl:element name="note">
      <xsl:attribute name="type">from</xsl:attribute>
```

[16] The bibliography part of this DTD comes frome the TEI. See http://www.tei-c.org/P4X/

```
      <xsl:value-of select="./@from"/>
    </xsl:element>
    <xsl:element name="note">
      <xsl:attribute name="type">userid</xsl:attribute>
      <xsl:value-of select="./@userid"/>
    </xsl:element>
  </xsl:element>
</xsl:template>
```

In order to make this document shorter, we have use the pseudo command "apply-many-templates" that calls apply-templates in order. The next three templates are specific to the TEI.

```
<xsl:template name="citation.mono">
  <xsl:element name="monogr">
    <xsl:apply-many-templates
      "btitle" and "bbooktitle" and "bseries" and "bjournal" and "bauteurs"
      and "beditеur" and "bnote" and "btype" and "bedition"/>
    <xsl:call-template name="imprint"/>
  </xsl:element>
</xsl:template>
```

This is a bit more complicated, because we have three parts `<analytic>`, `<monogr>` and `<imprint>`.

```
<xsl:template name="citation.analy">
  <xsl:element name="analytic">
    <xsl:apply-many templates "btitle" and "bauteurs"/>
  </xsl:element>
  <xsl:element name="monogr">
    <xsl:apply-many-templates "beditеur" and "bbooktitle" and "bseries"
      and "bjournal" and "bnote" and "btype"/>
    <xsl:call-template name="imprint"/>
  </xsl:element>
</xsl:template>
```

This is used to indicate how the reference is published.

```
<xsl:template name="imprint">
  <xsl:element name="imprint">
    <xsl:apply-many-templates "bvolume" and "bchapter" and "bnumber"
       and "bpublisher" and "bschool" and "borganization" and "binstitution"/>
    <xsl:call-template name="bdate"/>
    <xsl:apply-many-templates "bpages" and "xref"/>
  </xsl:element>
</xsl:template>
```

Transformation of the `<btitle>`. The result is a `<title>` element, whose attribute depends on the type. We have not shown the value of the second test. Using 'choose' and 'otherwise' would make the code more robust.

```
<xsl:template match="btitle">
  <xsl:if test="../@type='article' or ../@type='inbook' or
       ../@type='incollection' or ../@type='inproceedings' or ../@type='conference'">
    <title level="a"> <xsl:apply-templates /> </title>
  </xsl:if>
  <xsl:if test="  ... ">
    <title level="m"> <xsl:apply-templates /> </title>
  </xsl:if>
</xsl:template>
```

Transformation of <bauteurs>. The result is a <author>, containing the same list of authors. We translate all the <bpers>, and replace <etal> by a person whose first name is 'ETAL'.

```
<xsl:template match="bauteurs">
  <xsl:element name="author">
    <xsl:for-each select="bpers|etal">
      <xsl:choose>
        <xsl:when test="name()='etal'">
          <xsl:element name="persName">
            <xsl:element name="foreName"><xsl:text>ETAL</xsl:text></xsl:element>
          </xsl:element>
        </xsl:when>
        <xsl:otherwise> <xsl:call-template name="personne"/> </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
  </xsl:element>
</xsl:template>
```

Transformation of <beditor>. The result is a <editor>. What about Mister Etal?

```
<xsl:template match="bediteur">
  <xsl:element name="editor">
    <xsl:for-each select="bpers">
      <xsl:call-template name="personne"/>
    </xsl:for-each>
  </xsl:element>
</xsl:template>
```

Transformation of <bpers> in <persName>. The attributes nom and prenom are translated into <foreName> and <surname>.[17]

```
<xsl:template name="personne">
 <xsl:element name="persName">
  <xsl:element name="foreName"> <xsl:value-of select="@prenom"/> </xsl:element>
  <xsl:element name="surname">  <xsl:value-of select="@nom"/> </xsl:element>
 </xsl:element>
</xsl:template>
```

We replace <bseries> by <title>.

```
<xsl:template match="bseries">
  <title level="s"> <xsl:value-of select="."/>  </title>
</xsl:template>
```

Ditto for <bjournal>, with a different attribute value.

```
<xsl:template match="bjournal">
  <title level="j"> <xsl:value-of select="."/>  </title>
</xsl:template>
```

Ditto for <booktitle>. In some cases, we add the value of the <baddress>.

```
<xsl:template match="bbooktitle">
  <title level="m">
    <xsl:value-of select="."/>
    <xsl:if test="../baddress"> <xsl:text>, </xsl:text><xsl:value-of select="../baddress"/>
    </xsl:if>
  </title>
</xsl:template>
```

We replace <bnote> by <note>.

[17] What about part and junior? They should be added to <surname>.

```
<xsl:template match="bnote">
  <note type="bnote"> <xsl:value-of select="."/> </note>
</xsl:template>
```

Ditto for<btype>, with a different attribute value.

```
<xsl:template match="btype">
  <note type="typdoc"> <xsl:value-of select="."/> </note>
</xsl:template>
```

Ditto for <bhowpublished>, with a different attribute value.

```
<xsl:template match="bhowpublished">
 <note type="howpublished">  <xsl:value-of select="."/>  </note>
</xsl:template>
```

The transformation of <bschool> is <publisher>, with an <orgName> that contains the name of the school. It can be followed by the value of the <baddress> field of the entry.

```
<xsl:template match="bschool">
 <xsl:element name="publisher">
  <xsl:element name="orgName">
   <xsl:attribute name="type">school</xsl:attribute>
   <xsl:value-of select="."/>
  </xsl:element>
  <xsl:if test="../baddress"><xsl:apply-templates select="../baddress"/></xsl:if>
 </xsl:element>
</xsl:template>
```

Same idea here. But the implementation is different.

```
<xsl:template match="borganization">
  <xsl:element name="publisher">
    <xsl:element name="orgName">
      <xsl:attribute name="type">organisation</xsl:attribute>
      <xsl:value-of select="."/>
      <xsl:if  test="../baddress">
        <xsl:element name="address">
          <xsl:apply-templates select="../baddress"/>
        </xsl:element>
      </xsl:if>
    </xsl:element>
  </xsl:element>
</xsl:template>
```

Same idea here.

```
<xsl:template match="binstitution">
  <xsl:element name="publisher">
    <xsl:element name="orgName">
      <xsl:attribute name="type">institution</xsl:attribute>
      <xsl:value-of select="."/>
      <xsl:if  test="../baddress">
        <xsl:element name="address">
          <xsl:apply-templates select="../baddress"/>
        </xsl:element>
      </xsl:if>
    </xsl:element>
  </xsl:element>
</xsl:template>
```

Strange.

```
<xsl:template match="bedition">
  <xsl:element name="edition">
```

```
     <xsl:value-of select="./text()"/>
     <xsl:copy-of select="./*"/>
   </xsl:element>
</xsl:template>
```

Translation of a date into a <dateStruct>. We take the <bmonth> and <byear> and convert them into <month> and <year>. Nothing is done if both fields are missing.

```
<xsl:template name="bdate">
 <xsl:if test="string-length(bmonth|byear)>0">
  <xsl:element name="dateStruct">
   <xsl:if test="string-length(bmonth)>0">
    <xsl:element name="month"> <xsl:value-of select="bmonth"/> </xsl:element>
   </xsl:if>
   <xsl:if test="string-length(byear)>0">
     <xsl:element name="year"> <xsl:value-of select="byear"/> </xsl:element>
   </xsl:if>
  </xsl:element>
 </xsl:if>
</xsl:template>
```

Transformation of <bvolume> into a <biblScope>.

```
<xsl:template match="bvolume">
  <biblScope type="volume">  <xsl:value-of select="."/> </biblScope>
</xsl:template>
```

Transformation of <bchapter> into a <biblScope> with a different attribute value.

```
<xsl:template match="bchapter">
  <biblScope type="chapter">  <xsl:value-of select="."/> </biblScope>
</xsl:template>
```

Transformation of <bnumber> into a <biblScope> with a different attribute value.

```
<xsl:template match="bnumber">
  <biblScope type="number">  <xsl:value-of select="."/> </biblScope>
</xsl:template>
```

Transformation of <bpages> into a <biblScope> with a different attribute value.

```
<xsl:template match="bpages">
  <biblScope type="number">  <xsl:value-of select="."/> </biblScope>
</xsl:template>
```

Transformation of <bpublisher> into a <publisher> containing a <orgName>, and sometimes the address.

```
<xsl:template match="bpublisher">
  <xsl:element name="publisher">
    <xsl:element name="orgName">
      <xsl:value-of select="."/>
      <xsl:apply-templates select="../baddress"/>
    </xsl:element>
  </xsl:element>
</xsl:template>
```

Translation of <baddress> into a <address> containing a <addrLine>.

```
<xsl:template match="baddress">
 <xsl:if test="not(../bbooktitle)">
   <xsl:element name="address">
      <xsl:element name="addrLine"><xsl:value-of select="."/></xsl:element>
   </xsl:element>
 </xsl:if>
</xsl:template>
```

Unused.

```
<xsl:template match="biblio/citation/xref">
 ...
</xsl:template>
```

This is the end of the file.

```
</xsl:transform>
```

6.5 Converting the bibliography into HTML

We explain here what is in the file that converts the bibliography into HTML code. The input is the result of the translation (code shown in the section above) of what Tralics produces, using a TEI syntax. The full TEI is not yet implemented.

We declare the style sheet and all namespaces.

```
<xsl:stylesheet
  xmlns:tei="http://www.tei-c.org/ns/1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
  xmlns:m="http://www.w3.org/1998/Math/MathML"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:html="http://www.w3.org/1999/xhtml"
  exclude-result-prefixes="m html xlink tei">
```

We remove spaces in most elements.

```
<xsl:strip-space elements="biblStruct monogr analytic author editor
   title imprint address addrLine"/>
```

In order to make the following code a bit shorter, we replace '`note[@type='from']/text()`' with `$from` and '`note[@type='classification']/text()`' with `$type`. These two quantities are not defined by the TEI, but are used to sort the bibliography. We make a strong assumption: that `$from` is one of 'year' or 'refer' or 'foot'. Entries are sorted by putting 'refer' first and 'foot' last. We assign category 'd' and 'k' to these entries. The second assumption is that `$type` is one of 'article', 'book', 'booklet', 'inbook', 'incollection', 'inproceedings', 'manual', 'masterthesis', 'misc', 'phdthesis', 'proceedings' 'techreport', 'unpublished'. These values are taken from [6, page 763]. Note that bibtex defines 'masterthesis' and 'mastersthesis' as being equivalent; the companion lists the name with an 's', Tralics outputs the other one. We accept also 'conference', this is the same as 'inproceedings', 'manuel' (like 'manual', not considered by Tralics) and 'coursenotes'. All these entries are classified[18]: first book, then Ph.D. theses, then articles in journals, then articles in conferences, then technical reports, then anything else. Their type is a letter between 'e' and 'j'.

The following code is invalid XSLT, because variables are forbidden in `<xsl:key>`, but the idea is there. The construction '`1 div X`' is a bit strange. There is an example in [4], that explains that it returns 1 in case X is the boolean value true, and infinity otherwise, so that the substring contains one or zero characters. In any case, this associates to each entry a value, that happens to be a single letter, because at most one substring produces a non-empty string.

```
<xsl:key name="bibliotypes" match="//biblio/biblStruct" use="
 concat(
    substring('d', 1 div ($from='refer')),
    substring('e', 1 div ($from='year' and ($type='book' or
             $type='booklet' or  $type='proceedings'))),
    substring('f', 1 div ($from='year' and ($type='phdthesis'))),
    substring('g', 1 div ($from='year' and ($type='article' or
            $type='inbook' or $type='incollection'))),
    substring('h', 1 div  ($from='year' and ($type='inproceedings' or  $type='conference'))),
```

[18] in the sense of 'sorted'; secret documents should not appear in the Raweb.

```
     substring('i', 1 div ($from='year' and ($type='techreport' or
        $type='manuel' or $type='manual' or  $type='coursenotes'))),
     substring('j', 1 div ($from='year' and ($type='unpublished' or
        $type='misc' or  $type='masterthesis'))),
     substring('k', 1 div ($from='foot'))
     )" />
```

Remember that the structure 'bibliotypes' is computed before any global variable is set. As a consequence, we can safely put in a variable the number of entries of each category.

```
<xsl:variable name='d' select="count(key('bibliotypes', 'd'))"/>
<xsl:variable name='e' select="count(key('bibliotypes', 'e'))"/>
<xsl:variable name='f' select="count(key('bibliotypes', 'f'))"/>
<xsl:variable name='g' select="count(key('bibliotypes', 'g'))"/>
<xsl:variable name='h' select="count(key('bibliotypes', 'h'))"/>
<xsl:variable name='i' select="count(key('bibliotypes', 'i'))"/>
<xsl:variable name='j' select="count(key('bibliotypes', 'j'))"/>
<xsl:variable name='k' select="count(key('bibliotypes', 'k'))"/>
```

In order to simplify the code that follows, we show here the source of one of the sections. It is a template with three arguments, a type, for instance 'f', a title, and the number of entries before the first of this type.

```
   <xsl:call-template name="tri">
     <xsl:with-param name="str" select="'f'"/>
     <xsl:with-param name="title" select="'some funny title'"/>
     <xsl:with-param name="countPrevious" select="$d+$e"/>
   </xsl:call-template>
```

The sections are in order from d to k, and the titles are respectively: 'Major publications by the team in recent years', 'Doctoral dissertations and Habilitation theses', 'Books and Monographs', 'Articles in referred journals and book chapters', 'Publications in Conferences and Workshops', 'Internal Reports', 'Miscellaneous', 'Bibliography in notes'.

```
<xsl:template name="fill.the.bib">
 <dl>
   the 8 sections are here
 </dl>
</xsl:template>
```

What the code does is obvious: we consider all entries that are of type `$str`. If the list is empty, nothing is done. Otherwise, the `$title` is output in a `<h2>`. After that, we output all elements of the list, sorted by author's name. The template that outputs the entry takes a number: it is the index of the entry, the value of `$countPrevious` plus the index in the sorted list.

```
<xsl:template name="tri">
  <xsl:param name="str"/>
  <xsl:param name="title"/>
  <xsl:param name="countPrevious"/>
  <xsl:if test="key('bibliotypes',$str)[1]">
    <h2><xsl:value-of select='$title'/></h2>
    <xsl:for-each select="key('bibliotypes',$str)">
       <xsl:sort select="descendant::author[1]/persName[1]/surname/text()"/>
       <xsl:apply-templates select=".">
          <xsl:with-param name="pos">
            <xsl:value-of select="$countPrevious+position()"/>
          </xsl:with-param>
       </xsl:apply-templates>
    </xsl:for-each>
  </xsl:if>
</xsl:template>
```

This template exists in two versions: there is a version used for debugging, that emits a warning in the case where the entry types are wrong. This is the production version. It converts a `<biblStruct>` into a `<dt>`, that contains an anchor and the number of the reference, and a `<dd>` that contains the reference. A non trivial point concerns punctuation. The idea is that there is a period after the list of authors, and at the end of the citation. Each item is preceded by a comma, except the author list (that is at the start of an entry) or the title (that comes after the authors).

```
<xsl:template match="biblStruct">
  <xsl:param name="pos" />
  <dt class="bib">
    <a name="{./@id}">
      <xsl:text>[</xsl:text><xsl:value-of select="$pos"/><xsl:text>]</xsl:text>
    </a>
   </dt>
  <dd> <xsl:apply-templates/> <xsl:text>.</xsl:text> </dd>
  <xsl:text> </xsl:text>
</xsl:template>
```

Most entries are trivial: a comma and the content. The case of `<edition>` is special: the LaTeX companion says that the value should be something like 'Second'. There is no such problem for `<orgName>` or `<addrLine>`.

```
<xsl:template match="edition|orgName|addrLine">
  <xsl:text>, </xsl:text> <xsl:apply-templates/>
</xsl:template>
```

Translation of `<biblScope>`. This depends on the `type` attribute. In the case 'volume', we output something like "vol. 25".

```
<xsl:template match="biblScope[@type='volume']">
  <xsl:text>, vol. </xsl:text> <xsl:apply-templates/>
</xsl:template>
```

In the case 'chapter', we output something like "chap. 25".

```
<xsl:template match="biblScope[@type='chapter']">
  <xsl:text>, chap. </xsl:text> <xsl:apply-templates/>
</xsl:template>
```

In the case 'number', we output something like "n° 25".

```
<xsl:template match="biblScope[@type='number']">
  <xsl:text>, n</xsl:text><sup>o</sup><xsl:text> </xsl:text>
  <xsl:apply-templates/>
</xsl:template>
```

In the case of 'pages' we output something like "p. 10–30". Note the funny test: we add a 'p.' only if the text contains a dash or an en-dash.

```
<xsl:template match="biblScope[@type='pages']">
  <xsl:text>, </xsl:text>
  <xsl:if test="string-length(substring-before(., '-'))
                or string-length(substring-before(., '&#8211;'))>0">
     <xsl:text>p. </xsl:text>
  </xsl:if>
  <xsl:apply-templates/>
</xsl:template>
```

Translation of `<note>`. This is trivial, but used only if the `type` attribute is 'bnote' (a BiBTeX note), 'typdoc' (this could be the type of a report) or 'howpublished' (for a non-standard entry).

```
<xsl:template match="note[@type='howpublished' | @type='bnote' | @type='typdoc']">
    <xsl:text>, </xsl:text>
```

```
    <xsl:apply-templates/>
</xsl:template>
```

These notes are used for sorting, their printed value is empty.

```
<xsl:template match="note[@type='from' |@type='userid' | @type='classification']">
</xsl:template>
```

Translation of `<title>`. Is this really needed?

```
<xsl:template match="title[ancestor::biblio]">
   <xsl:text>"</xsl:text> <xsl:apply-templates/><xsl:text>"</xsl:text>
</xsl:template>
```

Translation of `<title>`. The attribute says that it is the name of a journal.

```
<xsl:template match="title[@level='j']">
  <xsl:text>, in: "</xsl:text> <xsl:apply-templates/><xsl:text>"</xsl:text>
</xsl:template>
```

Translation of `<title>`. The attribute says that it is the name of a series.

```
<xsl:template match="title[@level='s']">
  <xsl:text>, </xsl:text> <xsl:apply-templates/>
</xsl:template>
```

Translation of `<title>`. The attribute says that it is the title of an article, a book, or something like that. The result is a `<span>` with an attribute that says to use an italic font.

```
<xsl:template match="title[@level='a']">
   <span class="textit"> <xsl:apply-templates/> </span>
</xsl:template>
```

Translation of `<title>`. The attribute says that it is something different than the cases considered above. The TEI says something like: If the title appears directly enclosed within an `<analytic>` element, the level, if given, must be 'a'; if it appears directly enclosed within a `<monogr>` element, level must be 'm', 'j', or 'u'; when `<title>` is directly enclosed by `<series>`, level must be 's'. As a consequence, if the test is true, we have two titles (for instance a book title, and the title of a part of the book).

```
<xsl:template match="title[@level='m']">
 <xsl:choose>
  <xsl:when test="string-length(../../analytic/title) > 0">
      <xsl:text>, in: "</xsl:text><xsl:apply-templates/><xsl:text>"</xsl:text>
  </xsl:when>
  <xsl:otherwise>
     <span class="textit"> <xsl:apply-templates/>  </span>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

Translation of `<author>`. We consider all `<persName>` in the list, typesetting them one after the other with a comma as separator, and a period at the end. There is a `<br>` at the end of the list. A special case is when the first name is ETAL. Otherwise, we output the `<foreName>` and the `<surname>` in small caps (via a `<span>`).

```
<xsl:template match="author">
 <xsl:for-each select="persName">
   <xsl:choose>
     <xsl:when test="foreName='ETAL'">
        <small><xsl:text>et al.</xsl:text></small>
     </xsl:when>
     <xsl:otherwise>
        <xsl:value-of select="foreName"/>
        <xsl:text> </xsl:text>
```

```
        <span class="smallcap"><xsl:value-of select="surname"/></span>
        <xsl:call-template name="separateur.objet"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:for-each>
  <br/>
</xsl:template>
```

The translation of `<editor>` is similar. If an author and an editor are given, they are separated by a comma (remember that there is a line break after the authors). We add a `<br>` at the end in the case where no author is given. Editors are separated by commas, but at the end, we put a '(editor)' or '(editors)' remark.

```
<xsl:template match="editor">
 <xsl:if test="../..//author"> <xsl:text>, </xsl:text></xsl:if>
 <xsl:for-each select="persName">
   <xsl:choose>
     <xsl:when test="foreName='ETAL'">
       <small><xsl:text>et al.</xsl:text></small>
     </xsl:when>
     <xsl:otherwise>
       <xsl:value-of select="foreName"/>
       <xsl:text> </xsl:text>
       <span class="smallcap"><xsl:value-of select="surname"/></span>
       <xsl:call-template name="separateurED.objet"/>
     </xsl:otherwise>
   </xsl:choose>
 </xsl:for-each>
 <xsl:text> (editor</xsl:text><xsl:call-template name="pluriel-p">
      <xsl:with-param name="liste" select="persName" /></xsl:call-template>
 <xsl:text>).</xsl:text><xsl:if test="not(../../*/author)"><br/></xsl:if>
</xsl:template>
```

This seems to be useless.

```
<xsl:template match="bedition"> ? </xsl:template>
```

Case of a `<dateStruct>`. We output the `<month>` and the `<year>`.

```
<xsl:template match="dateStruct">
  <xsl:text>, </xsl:text>
  <xsl:value-of select="month"/><xsl:text> </xsl:text><xsl:value-of select="year"/>
</xsl:template>
```

Case of `<publisher>`. We output the two parts: `<orgName>` and optionally `<address>`.

```
<xsl:template match="publisher">
  <xsl:apply-templates select="orgName"/>
  <xsl:if test="./address"><xsl:apply-templates select="address"/> </xsl:if>
</xsl:template>
```

Case of `<address>`. No initial comma.

```
<xsl:template match="address">
  <xsl:apply-templates/>
</xsl:template>
```

Case of a reference. We assume that it is always an external reference. Moreover, the program that converts from the old DTD to the new one never adds a url attribute[19]. The essential difference between the two cases is that a `<br>` is added before the link.

[19]The value of the link is in xlink. It happens that Tralics puts the same value in the link and in the attribute in most of the cases. But this is liable to change.

```
<xsl:template match="biblio//ref">
  <xsl:choose>
   <xsl:when test="./@url">
     <a>
       <xsl:attribute name="target">_alt</xsl:attribute>
       <xsl:attribute name="href"><xsl:value-of select="./@url"/></xsl:attribute>
       <xsl:apply-templates/>
     </a>
   </xsl:when>
   <xsl:otherwise>
      <xsl:text>, </xsl:text>
      <br/>
      <a>
        <xsl:attribute name="target">_alt</xsl:attribute>
        <xsl:attribute name="href"><xsl:value-of select="."/></xsl:attribute>
        <xsl:apply-templates/>
      </a>
   </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Case of `<analytic>`. We output `<author>`, `<title>`, `<editor>`, and `<imprint>`. If no author is given the editor is put before the title.

```
<xsl:template match="analytic">
  <xsl:apply-templates select="author"/>
  <xsl:choose>
    <xsl:when test="author">
      <xsl:apply-templates select="title"/>
      <xsl:apply-templates select="editor"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:apply-templates select="editor"/>
      <xsl:apply-templates select="title"/>
    </xsl:otherwise>
  </xsl:choose>
 <xsl:apply-templates select="imprint"/>
</xsl:template>
```

Case of `<monogr>`. We output `<title>`, `<author>`, and `<editor>`, in some strange order, followed by `<note>`, `<edition>` and `<imprint>`.

```
<xsl:template match="monogr">
 <xsl:choose>
  <xsl:when test="../analytic">
    <xsl:apply-templates select="title[@level='m']"/>
    <xsl:apply-templates select="title[@level='j']"/>
    <xsl:apply-templates select="author"/>
    <xsl:apply-templates select="editor"/>
    <xsl:apply-templates select="title[@level='s']"/>
  </xsl:when>
  <xsl:otherwise>
    <xsl:apply-templates select="author"/>
    <xsl:apply-templates select="editor"/>
    <xsl:apply-templates select="title"/>
   </xsl:otherwise>
 </xsl:choose>
 <xsl:apply-templates select="note"/>
 <xsl:apply-templates select="edition"/>
```

```
 <xsl:apply-templates select="imprint"/>
</xsl:template>
```

Case of <imprint>. We output <publisher>, <dateStruct>, <biblScope>, and <ref>.

```
<xsl:template match="imprint">
  <xsl:apply-templates select="publisher"/>
  <xsl:apply-templates select="dateStruct"/>
  <xsl:apply-templates select="biblScope"/>
  <xsl:apply-templates select="ref"/>
</xsl:template>
```

This is the end of the file.

```
</xsl:stylesheet>
```

Chapter 7

Converting XML to XSL/Format

In this chapter, we consider conversion of XML to XSL/Format, using the old DTD. The files described in this chapter are part of the Tralics distribution.

7.1 The rrrafo3.xsl file

This file contains some commands that deal with fonts. In the original version version of Tralics, the translation of `{\tiny etc}` was a `<hi>` element, with an attribute rend=‘`small`’. In the current version, Tralics understands more than three sizes (small, large, and normal), and can indicate this in an element, rather than an attribute. Thus, the translation can be `<font-small4>`. In this file, we have a big template, that interprets all values of the rend attribute, and a lot of trivial, small templates, that interpret elements like `<small4>`.

This interprets the rend attribute according to the TEI, plus extension of the Raweb. The effect is to add attributes to the current element, in general `<fo:inline>`. The template takes two arguments `$defaultvalue` and `$defaultstyle`. They are used in case of unknown specification (the default is to use bold). Note: Perhaps, we should replace the value of ‘small’ by 9pt instead of 8pt.

```
<xsl:template name="rend">
  <xsl:param name="defaultvalue"/>
  <xsl:param name="defaultstyle"/>
  <xsl:choose>
    <xsl:when test="@rend='gothic'">
      <xsl:attribute name="font-family">cursive</xsl:attribute>
    </xsl:when>
    <xsl:when test="@rend='ital'">
      <xsl:attribute name="font-style">italic</xsl:attribute>
    </xsl:when>
    <xsl:when test="@rend='it'">
      <xsl:attribute name="font-style">italic</xsl:attribute>
    </xsl:when>
    <xsl:when test="@rend='slanted'">
      <xsl:attribute name="font-style">italic</xsl:attribute>
    </xsl:when>
    <xsl:when test="@rend='emph'">
      <xsl:attribute name="font-style">italic</xsl:attribute>
    </xsl:when>
    <xsl:when test="@rend='small'">
```

```
21        <xsl:attribute name="font-size">8pt</xsl:attribute>
22      </xsl:when>
23      <xsl:when test="@rend='small1'">
24        <xsl:attribute name="font-size">9pt</xsl:attribute>
25      </xsl:when>
26      <xsl:when test="@rend='small2'">
27        <xsl:attribute name="font-size">8pt</xsl:attribute>
28      </xsl:when>
29      <xsl:when test="@rend='small3'">
30        <xsl:attribute name="font-size">7pt</xsl:attribute>
31      </xsl:when>
32      <xsl:when test="@rend='small4'">
33        <xsl:attribute name="font-size">5pt</xsl:attribute>
34      </xsl:when>
35      <xsl:when test="@rend='large'">
36        <xsl:attribute name="font-size">12pt</xsl:attribute>
37      </xsl:when>
38      <xsl:when test="@rend='large1'">
39        <xsl:attribute name="font-size">12pt</xsl:attribute>
40      </xsl:when>
41      <xsl:when test="@rend='large2'">
42        <xsl:attribute name="font-size">14pt</xsl:attribute>
43      </xsl:when>
44      <xsl:when test="@rend='large3'">
45        <xsl:attribute name="font-size">17pt</xsl:attribute>
46      </xsl:when>
47      <xsl:when test="@rend='large4'">
48        <xsl:attribute name="font-size">20pt</xsl:attribute>
49      </xsl:when>
50      <xsl:when test="@rend='large5'">
51        <xsl:attribute name="font-size">25pt</xsl:attribute>
52      </xsl:when>
53      <xsl:when test="@rend='i'">
54        <xsl:attribute name="font-style">italic</xsl:attribute>
55      </xsl:when>
56      <xsl:when test="@rend='sc'">
57        <xsl:attribute name="font-variant">small-caps</xsl:attribute>
58      </xsl:when>
59      <xsl:when test="@rend='bo'">
60        <xsl:attribute name="font-weight">bold</xsl:attribute>
61      </xsl:when>
62      <xsl:when test="@rend='BO'">
63         <xsl:attribute name="font-style">italic</xsl:attribute>
64         <xsl:attribute name="text-decoration">underline</xsl:attribute>
65      </xsl:when>
66      <xsl:when test="@rend='UL'">
67        <xsl:attribute name="text-decoration">underline</xsl:attribute>
68      </xsl:when>
69      <xsl:when test="@rend='tt'">
70        <xsl:attribute name="font-family">Computer-Modern-Typewriter</xsl:attribute>
71      </xsl:when>
72      <xsl:when test="@rend='courier'">
73        <xsl:attribute name="font-family">iso-monospace</xsl:attribute>
74      </xsl:when>
75      <xsl:when test="@rend='sub'">
76        <xsl:attribute name="vertical-align">sub</xsl:attribute>
```

```
77      </xsl:when>
78      <xsl:when test="@rend='sup'">
79        <xsl:attribute name="vertical-align">super</xsl:attribute>
80      </xsl:when>
81      <xsl:when test="@rend='oldstyle'">
82        <xsl:attribute name="font-family">Concrete</xsl:attribute>
83      </xsl:when>
84      <xsl:otherwise>
85        <xsl:attribute name="{$defaultstyle}">
86          <xsl:value-of select="$defaultvalue"/>
87        </xsl:attribute>
88      </xsl:otherwise>
89    </xsl:choose>
90  </xsl:template>
```

Replacement code for the rend attribute defined above. In all cases, we provide a long and a short name. We produce an <fo:inline> element, with some attributes. We start with two templates associated to <small> and <large>. The attribute we set is font-size.

```
91  <xsl:template match='small|font-small'>
92    <fo:inline font-size='8pt'> <xsl:apply-templates/> </fo:inline>
93  </xsl:template>

94  <xsl:template match='large|font-large'>
95    <fo:inline font-size='12pt'> <xsl:apply-templates/> </fo:inline>
96  </xsl:template>
```

We continue with the ten font size commands of LaTeX.

```
97  <xsl:template match='small4|font-small4'>
98    <fo:inline font-size='5pt'> <xsl:apply-templates/> </fo:inline>
99  </xsl:template>

100 <xsl:template match='small3|font-small3'>
101   <fo:inline font-size='7pt'> <xsl:apply-templates/> </fo:inline>
102 </xsl:template>

103 <xsl:template match='small2|font_small2'>
104   <fo:inline font-size='8pt'> <xsl:apply-templates/> </fo:inline>
105 </xsl:template>

106 <xsl:template match='small1|font-small1'>
107   <fo:inline font-size='9pt'> <xsl:apply-templates/> </fo:inline>
108 </xsl:template>

109 <xsl:template match='normalsize|font-normalsize'>
110   <fo:inline font-size='10pt'> <xsl:apply-templates/> </fo:inline>
111 </xsl:template>

112 <xsl:template match='large1|font-large1'>
113   <fo:inline font-size='12pt'> <xsl:apply-templates/> </fo:inline>
114 </xsl:template>

115 <xsl:template match='large2|font-large2'>
116   <fo:inline font-size='14.4pt'> <xsl:apply-templates/> </fo:inline>
117 </xsl:template>

118 <xsl:template match='large3|font-large3'>
119   <fo:inline font-size='17.28pt'> <xsl:apply-templates/> </fo:inline>
120 </xsl:template>

121 <xsl:template match='large4|font-large4'>
122   <fo:inline font-size='20.74pt'> <xsl:apply-templates/> </fo:inline>
123 </xsl:template>
```

```
<xsl:template match='large5|font-large5'>
  <fo:inline font-size='24.88pt'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

Now the four shapes. The attribute is font-style or font-variant.

```
<xsl:template match='it|font-italic-shape'>
  <fo:inline font-style='italic'> <xsl:apply-templates/> </fo:inline>
</xsl:template>

<xsl:template match='slanted|font-slanted-shape'>
  <fo:inline font-style='oblique'> <xsl:apply-templates/>  </fo:inline>
</xsl:template>

<xsl:template match='sc|font-small-caps-shape'>
  <fo:inline font-variant='small-caps'> <xsl:apply-templates/> </fo:inline>
</xsl:template>

<xsl:template match='upright|font-upright-shape'>
  <fo:inline font-variant='normal'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

Now the three families. The attribute is font-variant or font-family. We provide two different tt families.

```
<xsl:template match='roman|font-roman-family'>
  <fo:inline font-variant='normal'> <xsl:apply-templates/> </fo:inline>
</xsl:template>

<xsl:template match='sansserif|font-sansserif-family'>
  <fo:inline font-family='sansserif'> <xsl:apply-templates/> </fo:inline>
</xsl:template>

<xsl:template match='computer-modern-tt'>
  <fo:inline font-family='Computer-Modern-Typewriter'>
    <xsl:apply-templates/>
  </fo:inline>
</xsl:template>

<xsl:template match='tt|font-typewriter-family'>
  <fo:inline font-family='monospace'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

Now the two series. The attribute is font-weight.

```
<xsl:template match='medium|font-medioum-series'>
  <fo:inline font-weight='medium'> <xsl:apply-templates/> </fo:inline>
</xsl:template>

<xsl:template match='bold|font-bold-series'>
  <fo:inline font-weight='bold'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

Superscript, subscript in text fonts. The attribute is vertical-align.

```
<xsl:template match='sup|font-super'>
  <fo:inline vertical-align='super'> <xsl:apply-templates/> </fo:inline>
</xsl:template>

<xsl:template match='sub|font-sub'>
  <fo:inline vertical-align='sub'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

Overline underline in text fonts. The attribute is text-decoration.

```
<xsl:template match='underline|font-underline'>
  <fo:inline text-decoration='underline'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
<xsl:template match='overline|font-overline'>
  <fo:inline text-decoration='overline'> <xsl:apply-templates/> /fo:inline>
</xsl:template>
```

Translation of <note>. Only footnotes are supported. The result is a <fo:footnote> element.

```
<xsl:template match="note">
  <xsl:variable name="FootID">
     <xsl:call-template name="calculateFootnoteNumber"/>
  </xsl:variable>
  <fo:footnote>
    <fo:inline font-size="{$footnotenumSize}" vertical-align="super">
      <xsl:value-of select="$FootID"/>
    </fo:inline>
    <fo:footnote-body>
      <fo:block end-indent="0pt" start-indent="0pt"
             text-indent="{$parIndent}" font-size="{$footnoteSize}">
        <fo:inline font-size="{$footnotenumSize}" vertical-align="super">
          <xsl:value-of select="$FootID"/>
        </fo:inline>
        <xsl:apply-templates/>
      </fo:block>
    </fo:footnote-body>
  </fo:footnote>
</xsl:template>
```

This is the end of the file.

```
</xsl:stylesheet>
```

7.2 The rawebfo file

This is the main file, its name is raweb3fo.xsl. It starts like this.

```
<xsl:stylesheet
  xmlns:fotex="http://www.tug.org/fotex"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
  xmlns:m="http://www.w3.org/1998/Math/MathML"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">
```

We overwrite some parameters defined in the TEI file.

```
<xsl:param name="linkColor">red</xsl:param>
<xsl:param name="pageMarginRight">80pt</xsl:param>
<xsl:param name="pdfBookmarks"></xsl:param>
```

We include these two files.

```
<xsl:include href="raweb3-param.xsl"/>
<xsl:include href="rrrafo3.xsl"/>
```

There are some elements for which white space is ignored.

```
<xsl:strip-space elements="cell bediteur bauteurs citation UR"/>
<xsl:output indent="no"/>
```

We do not use table specifications. Thus this code is a simplification of the original.

```
<xsl:variable name="top" select="/"/>
<xsl:variable name="tableSpecs">
  <Info></Info>
</xsl:variable>
```

In the case of unknown elements, we put the name in a comment, and we handle the content.

```
<xsl:template match="*">
  <xsl:comment><xsl:text>PASS THROUGH </xsl:text>
     <xsl:value-of select="name()"/>
  </xsl:comment>
  <xsl:apply-templates/>
</xsl:template>
```

7.2.1 Page definitions

The first object in the XSL/Format file is the definition of the page layout. In our case, half of the definitions given here are not used, but are taken from the TEI code. Maybe one day somebody wants an index in two columns, in this case, page masters with index 2 can be used. Currently, there is no index mechanism offered by Tralics.

The `<fo:layout-master-set>` element contains first some `<fo:simple-page-master>` elements. All these define the value `$pageMarginTop` for the top margin, `$pageMarginRight` for the bottom margin, `$pageMarginLeft` for the left margin, and `$pageMarginRight` for the right margin. We shall not indicate these values everywhere. Some define `$pageWidth` for the width of the page, and `$pageHeight` for the height of the page. In any case, we define three regions, `<fo:region-body>`, `<fo:region-before>`, and `<fo:region-after>`. The extent is always the same: `$regionBeforeExtent` and `$regionAfterExtent`.

We start with simple1: no headings, one column, all pages have the same look. The regions before and after have no name.

```
<xsl:template name="setupPagemasters">
  <fo:layout-master-set>
     <fo:simple-page-master
        master-name="simple1"
        page-width="{$pageWidth}"
        page-height="{$pageHeight}"
        margin-top="{$pageMarginTop}"
        margin-bottom="{$pageMarginBottom}"
        margin-left="{$pageMarginLeft}"
        margin-right="{$pageMarginRight}">
        <fo:region-body
                margin-bottom="{$bodyMarginBottom}"
                margin-top="{$bodyMarginTop}"/>
        <fo:region-before extent="{$regionBeforeExtent}"/>
        <fo:region-after extent="{$regionAfterExtent}"/>
     </fo:simple-page-master>
```

This is for left-hand/even pages in twosided mode, single column.

```
     <fo:simple-page-master
        master-name="left1"
        <!-- Attributes as above -->
        page-width="{$pageWidth}"
        page-height="{$pageHeight}">
        <fo:region-body
                margin-bottom="{$bodyMarginBottom}"
                margin-top="{$bodyMarginTop}"/>
```

```
237         <fo:region-before
238                 region-name="xsl-region-before-left"
239                 extent="{$regionBeforeExtent}"/>
240         <fo:region-after
241                 region-name="xsl-region-after-left"
242                 extent="{$regionAfterExtent}"/>
243       </fo:simple-page-master>
```

Case of right-hand/odd pages in twosided mode, single column.

```
244       <fo:simple-page-master
245         master-name="right1"
246         <!-- Attributes as above -->
247         page-width="{$pageWidth}"
248         page-height="{$pageHeight}">
249         <fo:region-body
250                 margin-bottom="{$bodyMarginBottom}"
251                 margin-top="{$bodyMarginTop}"/>
252         <fo:region-before
253                 region-name="xsl-region-before-right"
254                 extent="{$regionBeforeExtent}"/>
255         <fo:region-after
256                 region-name="xsl-region-after-right"
257                 extent="{$regionAfterExtent}"/>
258       </fo:simple-page-master>
```

Special case of first page in either mode, single column.

```
259       <fo:simple-page-master master-name="first1">
260         <!-- Attributes as above -->
261         <fo:region-body
262                 margin-bottom="{$bodyMarginBottom}"
263                 margin-top="{$bodyMarginTop}"/>
264         <fo:region-before
265                 region-name="xsl-region-before-first"
266                 extent="{$regionBeforeExtent}"/>
267         <fo:region-after
268                 region-name="xsl-region-after-first"
269                 extent="{$regionAfterExtent}"/>
270       </fo:simple-page-master>
```

Case of a blank page. No headings here.

```
271       <fo:simple-page-master master-name="blank1">
272         <!-- Attributes as above -->
273         <fo:region-body
274                 margin-bottom="{$bodyMarginBottom}"
275                 margin-top="{$bodyMarginTop}"/>
276         <fo:region-before
277                 region-name="DummyRegion"
278                 extent="{$regionBeforeExtent}"/>
279         <fo:region-after
280                 region-name="DummyRegion"
281                 extent="{$regionAfterExtent}"/>
282       </fo:simple-page-master>
```

For pages in one-side mode, 2 columns per page.

```
283       <fo:simple-page-master
284         master-name="simple2"
285         <!-- Attributes as above -->
286         page-width="{$pageWidth}"
```

```
      page-height="{$pageHeight}">
      <fo:region-body
              column-count="{$columnCount}"
              margin-bottom="{$bodyMarginBottom}"
              margin-top="{$bodyMarginTop}"/>
      <fo:region-before extent="{$regionBeforeExtent}"/>
      <fo:region-after extent="{$regionAfterExtent}"/>
   </fo:simple-page-master>
```

For left-hand/even pages in twosided mode, 2 columns per page.

```
  <fo:simple-page-master
     master-name="left2"
     <!-- Attributes as above -->
     page-width="{$pageWidth}"
     page-height="{$pageHeight}">
     <fo:region-body
             column-count="{$columnCount}"
             margin-bottom="{$bodyMarginBottom}"
             margin-top="{$bodyMarginTop}"/>
     <fo:region-before
             region-name="xsl-region-before-left"
             extent="{$regionBeforeExtent}"/>
     <fo:region-after
             region-name="xsl-region-after-left"
             extent="{$regionAfterExtent}"/>
   </fo:simple-page-master>
```

For right-hand/odd pages in twosided mode, 2 columns per page.

```
   <fo:simple-page-master
     master-name="right2"
     <!-- Attributes as above -->
     page-width="{$pageWidth}"
     page-height="{$pageHeight}" >
     <fo:region-body
             column-count="{$columnCount}"
             margin-bottom="{$bodyMarginBottom}"
             margin-top="{$bodyMarginTop}"/>
     <fo:region-before
             region-name="xsl-region-before-right"
             extent="{$regionBeforeExtent}"/>
     <fo:region-after
             region-name="xsl-region-after-right"
             extent="{$regionAfterExtent}"/>
  </fo:simple-page-master>
```

Special case of first page in either mode, two columns per page.

```
   <fo:simple-page-master master-name="first2">
     <!-- Attributes as above -->
     <fo:region-body
             column-count="{$columnCount}"
             margin-bottom="{$bodyMarginBottom}"
             margin-top="{$bodyMarginTop}"/>
     <fo:region-before
             region-name="xsl-region-before-first"
             extent="{$regionBeforeExtent}"/>
     <fo:region-after
             region-name="xsl-region-after-first"
```

```
338                 extent="{$regionAfterExtent}"/>
339       </fo:simple-page-master>
```

We define now some `<fo:page-sequence-master>` elements.
They contain a `<fo:repeatable-page-master-alternatives>` element. These contain some `<fo:conditional-page-master-reference>`. We start with setup for double-sided, 1 column, no first page.

```
340       <fo:page-sequence-master master-name="twoside1nofirst">
341        <fo:repeatable-page-master-alternatives>
342          <fo:conditional-page-master-reference
343                  master-reference="right1"
344                  odd-or-even="odd"/>
345          <fo:conditional-page-master-reference
346                  master-reference="left1"
347                  odd-or-even="even"/>
348        </fo:repeatable-page-master-alternatives>
349       </fo:page-sequence-master>
```

Setup for double-sided, 1 column.

```
350       <fo:page-sequence-master master-name="twoside1">
351        <fo:repeatable-page-master-alternatives>
352          <fo:conditional-page-master-reference
353                  master-reference="first1"
354                  page-position="first"/>
355          <fo:conditional-page-master-reference
356                  master-reference="right1"
357                  odd-or-even="odd"/>
358          <fo:conditional-page-master-reference
359                 master-reference="left1"
360                 odd-or-even="even"/>
361          <fo:conditional-page-master-reference
362                  master-reference="blank1"
363                  blank-or-not-blank="blank"/>
364        </fo:repeatable-page-master-alternatives>
365       </fo:page-sequence-master>
```

Setup for single-sided, 1 column.

```
366       <fo:page-sequence-master  master-name="oneside1">
367        <fo:repeatable-page-master-alternatives>
368          <fo:conditional-page-master-reference
369                  master-reference="first1"
370                  page-position="first"/>
371          <fo:conditional-page-master-reference master-reference="simple1"/>
372        </fo:repeatable-page-master-alternatives>
373       </fo:page-sequence-master>
```

Setup for double-sided, 2 columns.

```
374       <fo:page-sequence-master master-name="twoside2">
375        <fo:repeatable-page-master-alternatives>
376          <fo:conditional-page-master-reference
377                  master-reference="first2"
378                  page-position="first"/>
379          <fo:conditional-page-master-reference
380                  master-reference="right2"
381                  odd-or-even="odd"/>
382          <fo:conditional-page-master-reference
383                  master-reference="left2"
384                  odd-or-even="even"/>
```

```
385       </fo:repeatable-page-master-alternatives>
386      </fo:page-sequence-master>
```

Setup for single-sided, 2 columns.

```
387      <fo:page-sequence-master  master-name="oneside2">
388       <fo:repeatable-page-master-alternatives>
389         <fo:conditional-page-master-reference
390                 master-reference="first2"
391                 page-position="first"/>
392         <fo:conditional-page-master-reference master-reference="simple2" />
393       </fo:repeatable-page-master-alternatives>
394      </fo:page-sequence-master>
395     <xsl:call-template name="hookDefinepagemasters"/>
```

That was a long template! The XSLT processor replaces all the space characters and newline characters by a single space. The resulting element is printed on a single line; it has over 5000 characters. This is much larger than the 500 that appear in the TEX source; in fact, in the case of the apics Team, TEX used 15174 input buffer positions.

```
396     </fo:layout-master-set>
397 </xsl:template>
```

We define here two `<fo:static-content>` elements, for left and right pages. They contain a `<fo:block>` and a second block (is this really needed?) The inner block says text-indent=‘`0pt`’, because headings should not be indented. We set border-after-style to solid. In the original version, this did not work, and there was a `<pagestylehrule>` instead. On one end we have page numbers, on the other we have the name of the team or INRIA.

```
398 <xsl:template name="myheaders">
399  <fo:static-content flow-name="xsl-region-before-right">
400   <fo:block  text-align="justify" font-size="{$bodySize}">
401     <fo:block border-after-style="solid" text-indent="0pt">
402        <fo:inline font-style="italic"><xsl:value-of select="$PRID"/></fo:inline>
403        <fo:leader rule-thickness="0pt"/>
404        <fo:inline> <fo:page-number/> </fo:inline>
405     </fo:block>
406   </fo:block>
407  </fo:static-content>
408
409  <fo:static-content flow-name="xsl-region-before-left">
410   <fo:block text-align="justify" font-size="{$bodySize}">
411     <fo:block border-after-style="solid" text-indent="0pt">
412        <fo:inline>  <fo:page-number/> </fo:inline>
413        <fo:leader rule-thickness="0pt"/>
414        <fo:inline font-style="italic">
415               Activity Report INRIA <xsl:value-of select="$year"/>
416        </fo:inline>
417     </fo:block>
418   </fo:block>
419  </fo:static-content>
```

These are not used by XSL/Format.

```
420  <fo:static-content flow-name="xsl-region-before-first"/>
421  <fo:static-content flow-name="xsl-region-after-right"/>
422  <fo:static-content flow-name="xsl-region-after-left"/>
423  <fo:static-content flow-name="xsl-region-after-first"/>
424 </xsl:template>
```

The main text is in a page sequence, defined like this.

```
<xsl:template name="maintext">
  <fo:page-sequence
        format="1"
        text-align="justify"
        hyphenate="true"
        language="english"
        initial-page-number="1"
        master-reference="twoside1"
      >
      <fo:flow
         flow-name="xsl-region-body"
         font-family="{$bodyFont}"
         font-size="{$bodySize}">
        <xsl:call-template name="raweb.body"/>
      </fo:flow>
  </fo:page-sequence>
</xsl:template>
```

The title page is another page sequence, defined like this. The content will be given later.

```
<xsl:template match="accueil">
   <fo:page-sequence
        format="1"
        text-align="justify"
        hyphenate="true"
        language="EN"
        initial-page-number="1"
        master-reference="twoside1"
        force-page-count="end-on-even"
      >
     <fo:flow font-style="italic" font-family="{$bodyFont}">
       <xsl:template  name="accueil.body"/>
     </fo:flow>
   </fo:page-sequence>
</xsl:template>
```

```
<xsl:template name="myTOC">
  <fo:page-sequence
        format="1"
        text-align="justify"
        hyphenate="true"
        language="english"
        initial-page-number="1"
        master-reference="twoside1"
        force-page-count="end-on-even"
      >
   <fo:flow flow-name="xsl-region-body">
    <fo:block>
       <xsl:template  name="toc.body"/>
    </fo:block>
   </fo:flow>
  </fo:page-sequence>
</xsl:template>
```

7.2.2 The text

The document element is `<raweb>`. The first thing to do is construct the `<fo:layout-master-set>`, then all `<fo:static-content>`. After that, we have three parts: the title page, the table of contents, and the main text. Each of these parts start on a right page (an odd one, but the first page in each section is numbered one). Using `<cleardoublepage>` is a big hack.[1]

```
<xsl:template match="raweb">
 <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
   <xsl:call-template name="setupPagemasters"/>
   <xsl:call-template name="myheaders"/>
   <xsl:apply-templates select="accueil"/>
   <xsl:call-template name="myTOC"/>
   <xsl:call-template name="maintext"/>
 </fo:root>
</xsl:template>
```

The `maintext` template selects a page sequence and a flow (see above), then calls this template. The idea is trivial: we consider one section after the other.

```
<xsl:template name="raweb.body">
  <xsl:apply-templates select="composition"/>
  <xsl:apply-templates select="presentation"/>
  <xsl:apply-templates select="fondements"/>
  <xsl:apply-templates select="domaine"/>
  <xsl:apply-templates select="logiciels"/>
  <xsl:apply-templates select="resultats"/>
  <xsl:apply-templates select="contrats"/>
  <xsl:apply-templates select="international"/>
  <xsl:apply-templates select="diffusion"/>
  <xsl:apply-templates select="biblio"/>
</xsl:template>
```

This computes the start of a section. The result is a `<fo:block>`, containing the number and title of the section; these are found in attributes. This can call two templates that can parameterise the code. Note: there is a problem with the bookmarks. For this reason, we do not show the code.

```
<xsl:template name="secNumberedHeading">
  <fo:block keep-with-next.within-page="always">
    <xsl:variable name="divid">
       <xsl:call-template name="idLabel"/>
    </xsl:variable>
    <xsl:attribute name="id"> <xsl:value-of select="$divid"/> </xsl:attribute>
    <xsl:attribute name="text-align">start</xsl:attribute>
    <xsl:attribute name="font-family">
          <xsl:value-of select="$divFont"/>
    </xsl:attribute>
    <xsl:call-template name="setupDiv0"/>
    <xsl:call-template name="blockStartHook"/>
    <xsl:value-of select="@numero"/>
    <xsl:text>. </xsl:text>
    <xsl:value-of select="@titre"/>
    <xsl:if test="$pdfBookmarks='true'">
      <fotex:bookmark ... />
```

[1] It is removed in the 2005 version shown here.

```
512      </xsl:if>
513    </fo:block>
514  </xsl:template>
```

This returns an id. Question? do we really need to replace underscores by hyphens?

```
515  <xsl:template name="idLabel">
516   <xsl:choose>
517     <xsl:when test="@id"><xsl:value-of select="translate(@id,'_','-')"/> </xsl:when>
518     <xsl:otherwise><xsl:value-of select="generate-id()"/></xsl:otherwise>
519   </xsl:choose>
520  </xsl:template>
```

This is for a subsection. The title is in `<head>`. The number has to be computed.

```
521  <xsl:template name="NumberedHeading">
522    <xsl:param name="level"/>
523    <fo:block keep-with-next.within-page="always">
524      <xsl:variable name="divid">
525         <xsl:call-template name="idLabel"/>
526      </xsl:variable>
527      <xsl:attribute name="id">
528        <xsl:value-of select="$divid"/>
529      </xsl:attribute>
530      <xsl:attribute name="text-align">start</xsl:attribute>
531      <xsl:attribute name="font-family">
532         <xsl:value-of select="$divFont"/>
533      </xsl:attribute>
534      <xsl:choose>
535       <xsl:when test="$level=0"><xsl:call-template name="setupDiv0"/></xsl:when>
536       <xsl:when test="$level=1"><xsl:call-template name="setupDiv1"/></xsl:when>
537       <xsl:when test="$level=2"><xsl:call-template name="setupDiv2"/></xsl:when>
538       <xsl:when test="$level=3"><xsl:call-template name="setupDiv3"/></xsl:when>
539       <xsl:when test="$level=4"><xsl:call-template name="setupDiv4"/></xsl:when>
540      </xsl:choose>
541      <xsl:call-template name="blockStartHook"/>
542      <xsl:variable name="Number">
543        <xsl:if test="$numberHeadings and $numberDepth &gt; $level">
544          <xsl:call-template name="calculateNumber">
545            <xsl:with-param name="numbersuffix" select="$headingNumberSuffix"/>
546          </xsl:call-template>
547        </xsl:if>
548      </xsl:variable>
549      <xsl:value-of select="$Number"/>
550      <xsl:apply-templates mode="section" select="head"/>
551      <xsl:if test="$pdfBookmarks='true'"> <fotex:toto/>
552        <fotex:bookmark   ... />
553      </xsl:if>
554    </fo:block>
555  </xsl:template>
```

The translation of the `<composition>` section is trivial. We add the toplevel moreinfo before the text.

```
556  <xsl:template match="composition">
557     <xsl:call-template name="secNumberedHeading"/>
```

```
558     <xsl:apply-templates select="/raweb/moreinfo"/>
559     <xsl:apply-templates/>
560   </xsl:template>
```

Other sections. We emit a title, then the content of the element.

```
561   <xsl:template
562       match="presentation|fondements|domaine|logiciels|resultats|contrats
563     |international|diffusion">
564     <xsl:call-template name="secNumberedHeading"/>
565     <xsl:apply-templates/>
566   </xsl:template>
```

Modules. This is a subsection. The title is in `<head>`. In the case of dummy titles, nothing is printed.

```
567   <xsl:template match="module">
568    <xsl:if test="./head!='(Sans Titre)'">
569     <xsl:call-template name="NumberedHeading">
570       <xsl:with-param name="level" select="'1'"/>
571     </xsl:call-template>
572    </xsl:if>
573    <xsl:apply-templates/>
574   </xsl:template>
```

A section is equivalent to `<div0>`, a module to `<div1>`. We consider here divisions of level 2, 3 and 4.

```
575   <xsl:template match="div2|div3|div4">
576     <xsl:call-template name="NumberedHeading">
577       <xsl:with-param name="level">
578          <xsl:value-of select="substring-after(name(),'div')"/>
579       </xsl:with-param>
580     </xsl:call-template>
581     <xsl:apply-templates/>
582   </xsl:template>
```

7.2.3 The table of contents

The table of contents is formed of a a title, followed by all sections and subsections, translated in a special mode.

```
583   <xsl:template name="toc.body">
584         <fo:block font-size="14pt" text-align="center" font-weight="bold"
585             space-after="20pt">
586           <xsl:text>Table of contents</xsl:text>
587         </fo:block>
588         <xsl:for-each select="//composition|//presentation|//fondements|
589               //domaine|//logiciels|//resultats|//contrats|//international|
590                //diffusion|//biblio|//module|//div2|//div3">
591            <xsl:apply-templates mode="xtoc" select="(.)"/>
592         </xsl:for-each>
593   </xsl:template>
```

Translation of a section in the TOC. The result is a bold line, containing the section number (computed), its title (from the attribute), the page number, and a link to it.

```
<xsl:template mode="xtoc" match="composition|presentation|
    fondements|domaine|logiciels|resultats|contrats|international|
     diffusion|biblio">
  <xsl:variable name="tocindent">
        <xsl:value-of select="$div0Tocindent"/>
  </xsl:variable>
  <fo:block>
      <xsl:attribute name="font-weight">bold</xsl:attribute>
      <xsl:attribute name="text-indent">
       <xsl:value-of select="$tocindent"/>
      </xsl:attribute>
   <xsl:call-template name="sec.num"/>
   <xsl:text> </xsl:text>
   <fo:inline>
     <xsl:value-of select="@titre"/>
   </fo:inline>
    <fo:leader rule-thickness="0pt"/>
    <fo:inline color="{$linkColor}">
      <xsl:variable name="pagref">
        <xsl:call-template name="idLabel"/>
      </xsl:variable>
     <fo:basic-link internal-destination="{$pagref}">
      <fo:page-number-citation ref-id="{$pagref}"/>
     </fo:basic-link>
    </fo:inline>
  </fo:block>
</xsl:template>
```

A module in the TOC. If the title (from the `<head>` element) is empty, we do nothing, otherwise call some template that adds a line to the TOC. The parameter of the template is 1 (a module is equivalent to a `<div1>`).

```
<xsl:template mode="xtoc" match="module">
 <xsl:if test="head!='(Sans Titre)'">
   <xsl:call-template name="tocheading">
    <xsl:with-param name="level">
        <xsl:value-of select="1"/></xsl:with-param>
   </xsl:call-template>
 </xsl:if>
</xsl:template>
```

A division (`<div2>`, `<div3>`, or `<div4>`) in the TOC. Same as for a module, but the parameter of the template, the number, has to be computed. Moreover, an entry is inserted in the TOC even when the title is empty.

```
<xsl:template mode="xtoc" match="div2|div3|div4">
   <xsl:call-template name="tocheading">
    <xsl:with-param name="level">
        <xsl:value-of select="substring-after(name(),'div')"/></xsl:with-param>
   </xsl:call-template>
</xsl:template>
```

A division gives an entry in the TOC as follows. The result is a `<fo:block>`, with some indentation that depends on the level, plus a number, and a title (unless empty), leaders, then the page number, and a link.

```
<xsl:template name="tocheading">
  <xsl:param name="level"/>
  <xsl:variable name="tocindent">
   <xsl:choose>
    <xsl:when test="$level='0'">
         <xsl:value-of select="$div0Tocindent"/></xsl:when>
    <xsl:when test="$level='1'">
         <xsl:value-of select="$div1Tocindent"/></xsl:when>
    <xsl:when test="$level='2'">
         <xsl:value-of select="$div2Tocindent"/></xsl:when>
    <xsl:when test="$level='3'">
         <xsl:value-of select="$div3Tocindent"/></xsl:when>
    <xsl:when test="$level='4'">
         <xsl:value-of select="$div4Tocindent"/></xsl:when>
    <xsl:otherwise><xsl:value-of select="$div1Tocindent"/></xsl:otherwise>
   </xsl:choose>
  </xsl:variable>
  <fo:block>
      <xsl:attribute name="text-indent">
       <xsl:value-of select="$tocindent"/>
      </xsl:attribute>
  <xsl:variable name="Number">
    <xsl:if test="$numberHeadings and $numberDepth &gt; $level">
     <xsl:call-template name="calculateNumber">
       <xsl:with-param name="numbersuffix" select="$tocNumberSuffix"/>
     </xsl:call-template>
   </xsl:if>
   </xsl:variable>
   <xsl:value-of select="$Number"/>
   <xsl:text> </xsl:text>
 <xsl:if test="head!='(Sans Titre)'">
   <fo:inline>
      <xsl:apply-templates mode="tocsection" select="head"/>
   </fo:inline>
 </xsl:if>
    <fo:leader rule-thickness="0pt"/>
    <fo:inline color="{$linkColor}">
      <xsl:variable name="pagref">
        <xsl:call-template name="idLabel"/>
      </xsl:variable>
     <fo:basic-link internal-destination="{$pagref}">
      <fo:page-number-citation ref-id="{$pagref}"/>
     </fo:basic-link>
    </fo:inline>
  </fo:block>
</xsl:template>
```

A `<head>` in the TOC: the ID disappears (as well as all other attributes). Moreover, the content is evaluated in 'section' mode. All elements disappear (only text remains), unless specified otherwise.

```
<xsl:template match="head" mode="tocsection">
  <xsl:apply-templates mode="section"/>
</xsl:template>
```

We interpret math elements in a title as usual (the title could be "Encoding π in π_{pa}", guess what happens without the math?)

```
<xsl:template match="m:math" mode="section">
 <m:math>
  <xsl:copy-of select="@*"/>
  <xsl:apply-templates mode="math"/>
 </m:math>
</xsl:template>
```

A <head> in a section: the ID is in the result (why do we replace underscores by dashes?).

```
<xsl:template match="head" mode="section">
 <fo:inline>
  <xsl:if test=".!='(Sans Titre)'">
   <xsl:if test="@id">
    <xsl:attribute name="id">
     <xsl:value-of select="translate(@id,'_','-')"/>
    </xsl:attribute>
   </xsl:if>
   <xsl:apply-templates mode="section"/>
  </xsl:if>
 </fo:inline>
</xsl:template>
```

Is this needed? I don't know.

```
<xsl:template match="anchor" mode="section"></xsl:template>
```

7.2.4 The bibliography

Translation of the bibliography. There are eight optional parts. We consider them one after the other.

```
<xsl:template match="biblio">
   <xsl:call-template name="secNumberedHeading"/>
   <xsl:call-template name="biblioA"/>
   <xsl:call-template name='biblioBA'/>
   <xsl:call-template name='biblioBC'/>
   <xsl:call-template name='biblioBD'/>
   <xsl:call-template name='biblioBE'/>
   <xsl:call-template name='biblioBH'/>
   <xsl:call-template name='biblioBJ'/>
   <xsl:call-template name="biblioC"/>
</xsl:template>
```

This outputs the title of a subsection, for the bibliography. This title is in the $name parameter, the default value is never used.

```
<xsl:template name="biblioname">
  <xsl:param name="name">Unknown bibliography section</xsl:param>
  <fo:block font-weight='bold' font-size="14pt" space-before="5pt"
       keep-with-next='always'>
    <xsl:value-of select="$name"/>
  </fo:block>
</xsl:template>
```

Bibliography, part one. We select all entries that have a from attribute whose value is 'refer'. The title of the section is "Major publications by the team in recent years". If no entry matches, the translation is empty.

```
<xsl:template name="biblioA">
  <xsl:if test="citation[@from ='refer']">
    <xsl:call-template name="biblioname">
      <xsl:with-param name="name">  ... </xsl:with-param>
    </xsl:call-template>
    <xsl:for-each select="citation[@from ='refer']">
       <xsl:apply-templates select="(.)"/>
    </xsl:for-each>
  </xsl:if>
</xsl:template>
```

Bibliography, last part. We select all entries that have a from attribute whose value is 'foot'. The title of the section is "Bibliography in notes".

```
<xsl:template name="biblioC">
  <xsl:if test="citation[@from ='foot']">
   ...
  </xsl:if>
</xsl:template>
```

We select all entries that have a from attribute whose value is 'year' and type is book or booklet or proceedings. The title of the section is "Books and Monographs".

```
<xsl:template name='biblioBA'>
 <xsl:if test="..."> ...  </xsl:if>
</xsl:template>
```

We select all entries that have a from attribute whose value is 'year' and type is phdthesis. The title of the section is "Doctoral dissertations and "Habilitation" theses".

```
<xsl:template name='biblioBC'>
  <xsl:if test="..."> ...  </xsl:if>
</xsl:template>
```

We select all entries that have a from attribute whose value is 'year' and type is article or inbook or incollection. The title of the section is "Articles in referred journals and book chapters".

```
<xsl:template name='biblioBD'>
  <xsl:if test="..."> ...  </xsl:if>
</xsl:template>
```

We select all entries that have a from attribute whose value is 'year' and type is inproceedings or conference. The title of the section is "Publications in Conferences and Workshops".

```
<xsl:template name='biblioBE'>
  <xsl:if test="..."> ...  </xsl:if>
</xsl:template>
```

We select all entries that have a from attribute whose value is 'year' and type is manual or techreport or coursenotes. The title of the section is "Internal Reports".

```
<xsl:template name='biblioBH'>
  <xsl:if test="..."> ...  </xsl:if>
</xsl:template>
```

We select all entries that have a from attribute whose value is 'year' and type is unpublished or misc or masterthesis or mastersthesis. The title of the section is "Miscellaneous".

```
<xsl:template name='biblioBJ'>
  <xsl:if test="..."> ...  </xsl:if>
</xsl:template>
```

A <citation> produces a <fo:block>, with the same id. It contains the value of the key, in brackets, then the content of the element.

```
<xsl:template match="citation">
  <fo:block space-before="15pt" text-indent="-2em">
     <xsl:attribute name="id"><xsl:value-of select="@id"/></xsl:attribute>
     [<xsl:value-of select="@key"/>]
     <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

This produces a comma between elements, a period at the end.

```
<xsl:template name="separateur.objet">
 <xsl:choose>
  <xsl:when test="position()!=last()">, </xsl:when>
  <xsl:when test="position()=last()">.</xsl:when>
 </xsl:choose>
</xsl:template>
```

This is old code, not used anymore.

```
<xsl:template name="separateur.objet.spec"> ? </xsl:template>
```

The translation of <bvolume>foo</bvolume> is 'Bar foo', where 'Bar' is the name attribute of the element (it could be 'volume' or 'vol.'; it is defined by the DTD).

```
<xsl:template match='bvolume|bnumber|bpages|bchapter|bseries|bedition'>
  <xsl:value-of select="@bname"/>
  <xsl:text> </xsl:text><xsl:apply-templates/>
  <xsl:call-template name="separateur.objet"/>
</xsl:template>
```

The translation of <btitle> is the title in italics.

```
<xsl:template match='btitle'>
  <fo:inline font-style='italic'><xsl:apply-templates/>. </fo:inline>
</xsl:template>
```

The translation of <bjournal>foo</bjournal> is "in « foo »", or something like that.

```
<xsl:template match='bjournal|bbooktitle'>
  <xsl:text>in « </xsl:text><xsl:apply-templates/>
       <xsl:text> »</xsl:text>
  <xsl:call-template name="separateur.objet"/>
</xsl:template>
```

Translation of most bibliographic elements is trivial.

```
<xsl:template match='byear|bmonth|btype|bschool|bpublisher|
   bnote|borganization|binstitution|baddress|bhowpublished'>
  <xsl:apply-templates/>
  <xsl:call-template name="separateur.objet"/>
</xsl:template>
```

Translation of a <bdoi> element. This is a link to 'http://dx.doi.org/xxx'.

```
<xsl:template match='bdoi'>
  <xsl:value-of select="@bname"/><xsl:text> </xsl:text>
  <fo:basic-link color="{$linkColor}">
   <xsl:attribute
     name="external-destination">http://dx.doi.org/<xsl:value-of select="."/>
   </xsl:attribute>
```

```
792     <xsl:apply-templates/>
793   </fo:basic-link>
794    <xsl:call-template name="separateur.objet"/>
795 </xsl:template>
```

Translation of `<bauteurs>`. This is a list of `<bpers>`, with an optional `<etal>` (this one seems wrongly translated; it has a nom but no prenom). For each author, we output the first name, the particle, the last name.[2]

```
796 <xsl:template match ="bauteurs">
797   <fo:inline font-variant='small-caps'>
798      <xsl:for-each select="bpers|etal">
799         <xsl:value-of select="@prenom"/>
800         <xsl:text> </xsl:text>
801         <xsl:if test="@part">
802            <xsl:value-of select="@part"/>
803            <xsl:text> </xsl:text>
804         </xsl:if>
805         <xsl:value-of select="@nom"/>
806         <xsl:call-template name="separateur.objet"/>
807      </xsl:for-each>
808    </fo:inline>
809    <xsl:text> </xsl:text>
810 </xsl:template>
```

Same idea. A comma is put after each editor. After that, 'editor' or 'editors' is added at the end.

```
811 <xsl:template match ="bediteur">
812   <fo:inline font-variant='small-caps'>
813      <xsl:for-each select="bpers|etal">
814         <xsl:value-of select="@prenom"/>
815         <xsl:text> </xsl:text>
816         <xsl:if test="@part">
817            <xsl:text> </xsl:text>
818            <xsl:value-of select="@part"/>
819         </xsl:if>
820         <xsl:value-of select="@nom"/>
821         <xsl:text>, </xsl:text>
822      </xsl:for-each>
823    </fo:inline>
824    <xsl:choose>
825     <xsl:when test="count(bpers|etal) != 1">
826        <xsl:text>editors</xsl:text>
827     </xsl:when>
828     <xsl:otherwise>
829       <xsl:text>editor</xsl:text>
830     </xsl:otherwise>
831    </xsl:choose>
832   <xsl:call-template name="separateur.objet"/>
833 </xsl:template>
```

Transformation of `<cit>`. The result is a link to a bibliographical item.

[2] What about junior?

```
<xsl:template match="cit">
 <fo:basic-link color="{$linkColor}">
    <xsl:attribute name="internal-destination">
      <xsl:value-of select="ref/@target"/>
    </xsl:attribute>
    <xsl:text>[</xsl:text>
    <xsl:value-of select="id(ref/@target)/@key"/>
    <xsl:text>]</xsl:text>
  </fo:basic-link>
</xsl:template>
```

7.2.5 People

Translation of `<catperso>`. This is like a subsection, with a title (in `<head>`), we select all the `<pers>` children.

```
<xsl:template match="catperso">
  <fo:block space-before="3pt">
    <fo:block font-weight='bold' text-indent="-15pt">
        <xsl:value-of select="head"/></fo:block>
    <xsl:for-each select="pers">
        <fo:block> <xsl:call-template name="pers"/></fo:block>
    </xsl:for-each>
  </fo:block>
</xsl:template>
```

Like above, but the result is inline: we do not put `<pers>` in a block, but use commas as separators.

```
<xsl:template match="participants|participant|participante|participantes">
 <fo:block space-after.optimum="4pt">
   <fo:inline>
     <xsl:attribute name="font-weight">bold</xsl:attribute>
       <xsl:value-of select="@titre"/>
   </fo:inline>
   <fo:inline>
    <xsl:for-each select="pers">
       <xsl:call-template name="pers"/>
       <xsl:call-template name="separateur.objet"/>
    </xsl:for-each>
   </fo:inline>
 </fo:block>
</xsl:template>
```

Transformation of `<pers>`. This is easy: we have two attributes, first name and last name. Unless empty, the content is put in brackets.

```
<xsl:template name="pers">
    <xsl:value-of select="./@prenom"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="./@nom"/>
    <xsl:if test="not(normalize-space(string(.)) ='')">
       <xsl:text> [</xsl:text>
        <xsl:apply-templates/>
       <xsl:text>]</xsl:text>
```

```
875     </xsl:if>
876 </xsl:template>
```

7.2.6 References

The translation of `<xref>` (external reference) is a link, that points to the value of the url attribute. It has some color...

```
877 <xsl:template match="xref">
878  <fo:basic-link color="{$linkColor}">
879    <xsl:attribute name="external-destination">
880      <xsl:value-of select="@url"/>
881    </xsl:attribute>
882    <xsl:apply-templates/>
883  </fo:basic-link>
884 </xsl:template>
```

The same code is used in the case when the link is a `<citation>` element; but we add a separator (comma or period) after it.

```
885 <xsl:template match="citation/xref">
886  <fo:basic-link color="{$linkColor}">
887    <xsl:attribute name="external-destination">
888      <xsl:value-of select="@url"/>
889    </xsl:attribute>
890    <xsl:apply-templates/>
891  </fo:basic-link>
892  <xsl:call-template name="separateur.objet"/>
893 </xsl:template>
```

Translation of an internal link. There is an attribute target that says to what it points. The `<ref>` element is empty, the link is not: the value of the link element is obtained by evaluating the target in the 'xref' mode. This should give a number (or a sequence like 12.3.4 for a subsection). In the case of a table, figure, math expression, this should be the number of the table, figure, etc.; in the case of a division, it should be the number associated to the title.

```
894 <xsl:template match="ref">
895  <fo:basic-link color="{$linkColor}">
896   <xsl:attribute name="internal-destination">
897     <xsl:value-of select="translate(@target,'_','-')"/>
898   </xsl:attribute>
899   <xsl:apply-templates mode="xref" select="id(@target)" />
900   <xsl:apply-templates/>
901  </fo:basic-link>
902 </xsl:template>
```

In the case of a section, computing the number is trivial: we take it from the DTD.

```
903 <xsl:template
904     match="biblio|presentation|fondements|domaine|logiciels|resultats|
905       contrats|composition|international|diffusion" mode = 'xref'>
906   <xsl:value-of select = "@numero"/>
907 </xsl:template>
```

In the case of a subsection, computing the number is non trivial. The first component is the section number, followed by a dot.

```
<xsl:template name="sec.num">
   <xsl:value-of select =
         "ancestor-or-self::*[self::composition or self::presentation
          or self::fondements or self::domaine or self::logiciels or
          self::resultats or self::contrats or self::international or
            self::diffusion or self::biblio]/@numero"/>
  <xsl:text>.</xsl:text>
</xsl:template>
```

This computes the number associated to a division (module, div2, div3, and div4) by counting these things. It is used twice: when the division is typeset (more exactly, when its title is typeset), and when the division appears in a link.

```
<xsl:template name="calculateNumber">
   <xsl:param name="numbersuffix"/>
   <xsl:call-template name="sec.num"/>
   <xsl:number level="multiple" from="raweb"
       count="module|div2|div3|div4"/>
   <xsl:value-of select="$numbersuffix"/>
</xsl:template>
```

This calls the template shown above.

```
<xsl:template mode="xref" match="module|div2|div3|div4">
  <xsl:call-template name="calculateNumber"/>
</xsl:template>
```

It is not clear where the label of a section should be: on the division or its title. For this reason this rule is added. In a previous version of Tralics, `<anchor>` elements were generated. These are now obsolete.

```
<xsl:template match="head|anchor" mode ="xref">
   <xsl:call-template name="calculateNumber"/>
</xsl:template>
```

The idea is that every math formula (a `<formula>` element) that has an id is numbered.

```
<xsl:template match="formula" mode="xref">
  <xsl:number level = "any" count="formula[@id]"/>
</xsl:template>
```

In the case of a figure, we count only figures that are not inline, via the rend attribute.

```
<xsl:template name="calculateFigureNumber">
  <xsl:number count="figure[@rend != 'inline']" level="any"/>
</xsl:template>

<xsl:template match='figure' mode="xref">
   <xsl:call-template name="calculateFigureNumber"/>
</xsl:template>
```

Same for a table.

```
<xsl:template name="calculateTableNumber">
    <xsl:number count="table[@rend != 'inline']" level="any"/>
</xsl:template>

<xsl:template match='table' mode="xref">
   <xsl:call-template name="calculateTableNumber"/>
</xsl:template>
```

In the case of a `<note>`, we count all elements that have a place attribute with value 'foot'.

```
<xsl:template name="calculateFootnoteNumber">
  <xsl:number  level="any" count="note[@place='foot']"/>
</xsl:template>
```

The case of an item is more complicated. Originally, there was a test: is the parent a `<list>` of type bibliography? in this case, '`[25]`' is produced instead of '`25`'. This test was removed. Thus, the only non trivial point is that numbering depends on the list level.

```
<xsl:template match="item" mode="xref">
  <xsl:variable name="listdepth" select="count(ancestor::list)"/>
  <xsl:variable name="listNFormat">
   <xsl:choose>
    <xsl:when test="$listdepth=1"> <xsl:text>1</xsl:text> </xsl:when>
    <xsl:when test="$listdepth=2"> <xsl:text>i</xsl:text> </xsl:when>
    <xsl:when test="$listdepth=3"> <xsl:text>a</xsl:text> </xsl:when>
    <xsl:when test="$listdepth=4"> <xsl:text>I</xsl:text> </xsl:when>
   </xsl:choose>
  </xsl:variable>
  <xsl:number format="{$listNFormat}"/>
</xsl:template>
```

7.2.7 Generic elements

A `<p>` element is translated into a `<fo:block>` element. It has some attributes, for instance a constant font size.

```
<xsl:template match="p">
  <fo:block font-size="{$bodySize}">
```

If the preceding sibling is a `<p>` element, and the current element has not noindent='`true`' as attribute, the paragraph will be indented, said otherwise, it will have a non-zero text-indent. The space-before attribute is computed as follows. We define a maximum value in the case where the preceding sibling is a `<p>`. We define an optimum value in the case where spacebefore is given as attribute to the current element. We define an optimum value if nothing is given, and the preceding sibling is a `<p>`. Note: we should always define all three values, because the default is zero, and this is wrong (but fotex interprets badly these quantities...)

```
    <xsl:if test="preceding-sibling::p">
       <xsl:if test ="not(@noindent)">
           <xsl:attribute name="text-indent">
              <xsl:value-of select="$parIndent"/>
           </xsl:attribute>
       </xsl:if>
       <xsl:choose>
         <xsl:when test="@spacebefore">
           <xsl:attribute name="space-before.optimum">
               <xsl:value-of select="@spacebefore"/>
           </xsl:attribute>
         </xsl:when>
         <xsl:otherwise>
           <xsl:attribute name="space-before.optimum">
              <xsl:value-of select="$parSkip"/>
           </xsl:attribute>
         </xsl:otherwise>
       </xsl:choose>
       <xsl:attribute name="space-before.maximum">
         <xsl:value-of select="$parSkipmax"/>
```

```
981         </xsl:attribute>
982       </xsl:if>
983       <xsl:if test="not(preceding-sibling::p)">
984          <xsl:if test="@spacebefore">
985             <xsl:attribute name="space-before.optimum">
986                <xsl:value-of select="@spacebefore"/>
987             </xsl:attribute>
988          </xsl:if>
989       </xsl:if>
```

In the case where the rend attribute is 'centered' or 'center', we set text-align to 'center'. If it is 'flushed-left' or 'flushed-right' we set it to left or right. If it is 'quoted' or 'justify' we set it to 'justify' (in the case 'quoted', we also set both left and right margins to 1 cm).

```
990       <xsl:choose>
991         <xsl:when test="@rend ='centered'">
992          <xsl:attribute name="text-align">center</xsl:attribute>
993         </xsl:when>
994         <xsl:when test="@rend ='center'">
995          <xsl:attribute name="text-align">center</xsl:attribute>
996         </xsl:when>
997         <xsl:when test="@rend ='flushed-left'">
998          <xsl:attribute name="text-align">left</xsl:attribute>
999         </xsl:when>
1000        <xsl:when test="@rend ='flushed-right'">
1001         <xsl:attribute name="text-align">right</xsl:attribute>
1002        </xsl:when>
1003        <xsl:when test="@rend ='quoted'">
1004         <xsl:attribute name="text-align">justify</xsl:attribute>
1005         <xsl:attribute name="margin-left">1cm</xsl:attribute>
1006         <xsl:attribute name="margin-right">1cm</xsl:attribute>
1007        </xsl:when>
1008        <xsl:when test="@rend ='justify'">
1009         <xsl:attribute name="text-align">justify</xsl:attribute>
1010        </xsl:when>
1011      </xsl:choose>
```

We insert the content of the <p> here.

```
1012      <xsl:apply-templates/>
1013    </fo:block>
1014  </xsl:template>
```

Conversion of <hi>. The result is an inline object, the non trivial part concerns transformation of the rend attribute, but this is defined elsewhere.

```
1015  <xsl:template match="hi">
1016    <fo:inline>
1017       <xsl:call-template name="rend">
1018         <xsl:with-param name="defaultvalue" select="string('bold')"/>
1019         <xsl:with-param name="defaultstyle" select="string('font-weight')"/>
1020       </xsl:call-template>
1021       <xsl:apply-templates/>
1022    </fo:inline>
1023  </xsl:template>
```

This is currently unused.

```
1024  <xsl:template match="code">
1025    <fo:inline font-family="{$typewriterFont}">
```

```
    <xsl:apply-templates/>
  </fo:inline>
</xsl:template>
```

This is currently unused.

```
<xsl:template match="ident">
  <fo:inline color="{$identColor}" font-family="{$sansFont}">
    <xsl:apply-templates/>
  </fo:inline>
</xsl:template>
```

This is trivial.

```
<xsl:template match="term">
  <fo:inline font-style="italic">
    <xsl:apply-templates/>
  </fo:inline>
</xsl:template>
```

7.2.8 Lists

This translates a `<list>`. If the first child is `<head>` we start with a block containg the title in italics. This is not used in the Raweb. In any case, the result is a `<fo:list-block>` with some constant right margin and a left margin that depends on the context: normal list, normal glossary, or glossary in a list. All items in the list are considered.

```
<xsl:template match="list">
 <xsl:if test="child::head">
  <fo:block font-style="italic"
        text-align="start"
        space-before.optimum="4pt">
    <xsl:apply-templates select="head"/>
  </fo:block>
 </xsl:if>
 <fo:list-block margin-right="{$listRightMargin}">
  <xsl:call-template name="setListIndents"/>
  <xsl:choose>
   <xsl:when test="@type='gloss'">
    <xsl:attribute name="margin-left">
     <xsl:choose>
     <xsl:when test="ancestor::list"><xsl:value-of
                        select="$listLeftGlossInnerIndent"/></xsl:when>
     <xsl:otherwise><xsl:value-of select="$listLeftGlossIndent"/></xsl:otherwise>
     </xsl:choose>
    </xsl:attribute>
  </xsl:when>
   <xsl:otherwise>
     <xsl:attribute name="margin-left">
         <xsl:value-of select="$listLeftIndent"/></xsl:attribute>
   </xsl:otherwise>
  </xsl:choose>
  <xsl:apply-templates select="item"/>
  </fo:list-block>
</xsl:template>
```

A list has also a space-before and space-after attributes, that depend on the level. The complete code is not shown here.

```
<xsl:template name="setListIndents">
  <xsl:variable name="listdepth" select="count(ancestor::list)"/>
   <xsl:choose>
     <xsl:when test="$listdepth=0">
         <xsl:attribute name="space-before">
            <xsl:value-of select="$listAbove-1"/>
         </xsl:attribute>
         <xsl:attribute name="space-after">
              <xsl:value-of select="$listBelow-1"/>
         </xsl:attribute>
     </xsl:when>
    <!-- Same for listdepth=1 2 or 3 -->
  </xsl:choose>
</xsl:template>
```

We use a template for this.

```
<xsl:template match="item">
  <xsl:call-template name="makeItem"/>
</xsl:template>
```

Translation of an `<item>`. The result is a `<fo:list-item>` that contains two sub-elements, we start with the first, the `<fo:list-item-label>`. It depends on the `type` attribute of our parent: simple, bullets, ordered, gloss, unordered. In any case, if we have an id, we associate it to the element; then we construct a `<fo:block>`. The case where the list is in the bibliography is not shown here. The case where the label has an `n` attribute is not shown here. In the case where the list has `type`=‘`ordered`’, we evaluate the item in xref mode: this produces a number; this number will be flushed right (alignment end). In the case where the list is a glossary, we will left align the label using a bold font; here the label is either a `<label>` child or a label sibling. In all other cases, if there is a label sibling it will be used. In both these cases, the label is transformed using a special mode.

```
<xsl:template name="makeItem">
 <xsl:variable name="listdepth" select="count(ancestor::list)"/>
 <fo:list-item space-before.optimum="{$listItemsep}">
  <fo:list-item-label>
     <xsl:if test="@id">
       <xsl:attribute name="id"><xsl:value-of select="@id"/></xsl:attribute>
    </xsl:if>
    <fo:block>
      <xsl:attribute name="margin-right">2.5pt</xsl:attribute>
      <xsl:choose>
        <xsl:when test="../@type='ordered'">
          <xsl:attribute name="text-align">end</xsl:attribute>
          <xsl:apply-templates mode="xref" select="."/>
          <xsl:text>.</xsl:text>
        </xsl:when>
        <xsl:when test="../@type='gloss'">
          <xsl:attribute name="text-align">start</xsl:attribute>
          <xsl:attribute name="font-weight">bold</xsl:attribute>
          <xsl:choose>
            <xsl:when test="label">
              <xsl:apply-templates mode="print" select="label"/>
            </xsl:when>
            <xsl:otherwise>
```

```
            <xsl:apply-templates mode="print" select="preceding-sibling::*[1]"/>
           </xsl:otherwise>
         </xsl:choose>
       </xsl:when>
        <xsl:when test="name(preceding-sibling::*[1])='label'">
         <xsl:apply-templates mode="print" select="preceding-sibling::*[1]"/>
        </xsl:when>
```

In all other cases, a default value is chosen, depending on the list level.

```
       <xsl:otherwise>
          <xsl:attribute name="text-align">center</xsl:attribute>
         <xsl:choose>
           <xsl:when test="$listdepth=1">
             <xsl:value-of select="$bulletOne"/>
           </xsl:when>
           <xsl:when test="$listdepth=2">
             <xsl:value-of select="$bulletTwo"/>
           </xsl:when>
           <xsl:when test="$listdepth=3">
             <xsl:value-of select="$bulletThree"/>
           </xsl:when>
           <xsl:when test="$listdepth=4">
             <xsl:value-of select="$bulletFour"/>
           </xsl:when>
         </xsl:choose>
       </xsl:otherwise>
     </xsl:choose>
   </fo:block>
 </fo:list-item-label>
```

Essentially, the translation is a `<fo:list-item-body>` that contains a `<fo:block>`. This block is contructed implicitely by a `<p>` or explicitely, with a normal weight for the font.

```
 <fo:list-item-body>
   <xsl:choose>
     <xsl:when test="p"> <xsl:apply-templates/> </xsl:when>
     <xsl:otherwise>
       <fo:block font-weight="normal"><xsl:apply-templates/></fo:block>
     </xsl:otherwise>
   </xsl:choose>
 </fo:list-item-body>
</fo:list-item>
</xsl:template>
```

When we have a `<label>`, this is associated to an item in a list.

```
<xsl:template mode="print" match="label">
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="label"/>
```

7.2.9 Images

This converts an image, or something like that. The result is a `<fo:external-graphic>` object. The src attribute is taken to be the file attribute. Initially, there was a `$graphicsPrefix` prefix

before the image; this was removed: images should be in the current directory. We consider the three attributes scale, width, height and angle. Let's hope that angle is not used: in TeX, putting the angle to ninety degrees exchanges the role of width and height, but you can specify the width after rotation. Since the order of attributes is irrelevant in XML, you lose[3]. But you cannot turn images in HTML. Note: if you specify a scale, you cannot specify a width nor a height. We removed a test to `$autoScaleFigures`, placed after all other tests.

```
<xsl:template name="generate-graphics">
  <fo:external-graphic src="{@file}">
    <xsl:choose>
      <xsl:when test="@scale">
         <xsl:attribute name="content-width">
           <xsl:value-of select="@scale * 100"/><xsl:text>%</xsl:text>
         </xsl:attribute>
      </xsl:when>
      <xsl:when test="@width">
        <xsl:attribute name="content-width">  <xsl:value-of select="@width"/>
        </xsl:attribute>
        <xsl:if test="@height">
          <xsl:attribute name="content-height">
            <xsl:value-of select="@height"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:if test="@angle">
           <xsl:attribute name="angle">
               <xsl:value-of select="@angle"/>
           </xsl:attribute>
        </xsl:if>
      </xsl:when>
      <xsl:when test="@height">
         <xsl:attribute name="content-height">
           <xsl:value-of select="@height"/>
         </xsl:attribute>
         <xsl:if test="@angle">
           <xsl:attribute name="angle">
             <xsl:value-of select="@angle"/>
           </xsl:attribute>
         </xsl:if>
      </xsl:when>
    </xsl:choose>
  </fo:external-graphic>
</xsl:template>
```

A `<figure>` element, that has rend=‘`inline`’, produces a graphic object by the routine shown above. If it has a head, this is a caption, shown after the image.

```
<xsl:template match="figure[@rend='inline']">
  <xsl:call-template name='generate-graphics'/>
  <xsl:if test="head">
     <fo:block text-align="center">
           <xsl:apply-templates select="head"/>
     </fo:block>
```

[3]This is a bug in Tralics, of course

```
  </xsl:if>
</xsl:template>
```

All other images produce a `<fo:float>` object. It has an id, and contains an `<fo:block>`, this block is created by the procedure shown above, in case there is a file attribute, or by evaluating all `<p>` children. The content is centered. There is a second block, that contains the caption (from `<head>`).

```
<xsl:template match='figure'>
 <fo:float>
   <xsl:call-template name="addID"/>
   <fo:block text-align="center">
    <xsl:choose>
      <xsl:when test="@file">
       <xsl:call-template name='generate-graphics'/>
      </xsl:when>
      <xsl:otherwise><xsl:apply-templates select='p'/></xsl:otherwise>
    </xsl:choose>
   </fo:block>
   <fo:block>
     <xsl:call-template name="figureCaptionstyle"/>
     <xsl:value-of select="$figureWord"/>
     <xsl:call-template name="calculateFigureNumber"/>
     <xsl:text>. </xsl:text>
     <xsl:apply-templates select="head"/>
   </fo:block>
 </fo:float>
</xsl:template>
```

7.2.10 Tables

Essentially, the table is handled by 'blockTable'; this produces a `<fo:table>` element; but if the table is not marked inline, we use 'floatTable', this will produce a `<fo:table-and-caption>`.

```
<xsl:template match="table">
 <xsl:choose>
  <xsl:when test="@rend='inline'"><xsl:call-template name="blockTable"/></xsl:when>
  <xsl:otherwise> <xsl:call-template name="floatTable"/> </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

This produces a `<fo:table-and-caption>` containing a table and a caption. The table is computed as in the case without caption. There is a mechanism that turns the whole table-and-caption by ninety degree, this is not used by Tralics. The caption is preceded by something like 'Figure 99', and the number has to be computed.

```
<xsl:template name="floatTable">
  <fo:table-and-caption>
      <xsl:if test="rend='landscape'">
        <xsl:attribute name="reference-direction">-90</xsl:attribute>
      </xsl:if>
      <xsl:call-template name="addID"/>
      <fo:table-caption>
        <fo:block text-align="{$tableCaptionAlign}"
              space-after="{$spaceBelowCaption}">
```

```
            <xsl:value-of select="$tableWord"/>
            <xsl:call-template name="calculateTableNumber"/>
            <xsl:text>. </xsl:text>
            <xsl:apply-templates select="head"/>
          </fo:block>
        </fo:table-caption>
        <xsl:call-template name="blockTable"/>
    </fo:table-and-caption>
</xsl:template>
```

Transforming a table is easy: we select every row, and every cell in the rows. There is however a non trivial point: the current fotex implementation assumes that columns widthes are given; hence we must compute them. Note: there is a `<xsl:text>` element, a contains a newline character; this improves the layout of the resulting XML; it seems useless.

```
<xsl:template name="blockTable">
 <fo:table  text-align="{$tableAlign}">
   <xsl:call-template name="addID"/>
   <xsl:call-template name="deriveColSpecs"/>
   <fo:table-body text-indent="0pt">
     <xsl:for-each select="row">
       <xsl:text></xsl:text> <!-- this is a newline character -->
       <fo:table-row>
         <xsl:apply-templates select="cell"/>
       </fo:table-row>
      </xsl:for-each>
   </fo:table-body>
 </fo:table>
</xsl:template>
```

This seems useless. One can however imagine the situation where the table specifications are stored somewhere and we need a unique identifier for the table.

```
<xsl:template name="generateTableID">
 <xsl:choose>
  <xsl:when test="@id"> <xsl:value-of select="@id"/> </xsl:when>
  <xsl:otherwise><xsl:text>Table-</xsl:text><xsl:number level='any'/>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

The procedure that computes the table specifications is in another file.

```
<xsl:template name="deriveColSpecs" >
  <xsl:variable name="no"> <xsl:call-template name="generateTableID"/>
  </xsl:variable>
  <xsl:call-template name="calculateTableSpecs"/>
</xsl:template>
```

7.2.11 Mathematics

This is the same as code given elsewhere. We leave the math unchanged.

```
<xsl:template match="m:math">
 <m:math>
  <xsl:copy-of select="@*"/>
  <xsl:apply-templates mode="math"/>
```

```
 </m:math>
</xsl:template>
<xsl:template match="m:*|@*|comment()|processing-instruction()|text()" mode="math">
  ...
</xsl:template>
```

We replace a formula by its content.

```
<xsl:template match="formula">
  <xsl:apply-templates/>
</xsl:template>
```

A special case is when the formula has 'display' as type. If there is an id, then the result is a <fotex:equation>, otherwise it is a <fotex:displaymath>. The content could be simpler. In the case of an equation, we use a <fo:inline> (why not use it in every case? or add the id to the equation ?)

```
<xsl:template match="formula[@type='display']">
 <xsl:choose>
    <xsl:when test="@id">
     <fo:inline>
      <xsl:attribute name="id"><xsl:value-of select="@id"/></xsl:attribute>
     <fotex:equation>
    <xsl:for-each select="m:math">
     <xsl:apply-templates mode="math"/>
    </xsl:for-each>
    </fotex:equation></fo:inline>
  </xsl:when>
  <xsl:otherwise>
   <fotex:displaymath>
    <xsl:for-each select="m:math">
     <xsl:apply-templates mode="math"/>
    </xsl:for-each>
   </fotex:displaymath>
   </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

We convert a <simplemath> element to a normal math element.

```
<xsl:template match="simplemath">
  <m:math><m:mi><xsl:apply-templates/></m:mi></m:math>
</xsl:template>
```

7.2.12 Other elements

Translation of <head>. We look at the parent. If is a division (it starts with 'div', or is a <module>) we do nothing (already done), otherwise, we just output it.

```
<xsl:template match="head" priority="10">
  <xsl:variable name="parent" select="name(..)"/>
  <xsl:if test="not(starts-with($parent,'div')) and not($parent = 'module')">
    <xsl:apply-templates/>
  </xsl:if>
</xsl:template>
```

We copy all <?xmltex?> instructions.

```
<xsl:template match="processing-instruction()[name()='xmltex']" >
 <xsl:message>xmltex pi <xsl:value-of select="."/></xsl:message>
 <xsl:copy-of select="."/>
</xsl:template>
```

Is this needed?

```
<xsl:variable name="processor">
   <xsl:value-of select="system-property('xsl:vendor')"/>
</xsl:variable>
```

This is defined like in other style sheets.

```
<xsl:variable name="LeTypeProjet">
   <!-- see elsewhere -->
</xsl:variable>

<xsl:variable name="year">
  <!-- see elsewhere -->
</xsl:variable>
```

This could be: 'Team Foo'.

```
<xsl:variable name="PRID">
 <xsl:value-of select="$LeTypeProjet"/>
 <xsl:text> </xsl:text>
 <xsl:value-of select="/raweb/accueil/projet"/>
</xsl:variable>
```

This creates the first page. We start with a `<fo:INRIA>` special element, giving it the current year as `year` attribute. This will insert the logo. After that, we have a link to the Team's home page, with the short and long name. We have a link to each UR. We have also a special element `<fo:RATHEME>` that completes our page hacking.

```
<xsl:template  name="accueil.body"/>
   <fo:INRIA year="{$year}"/>
   <fo:block font-size= "25pt" text-align="center">
     <fo:basic-link external-destination="http://www.inria.fr/recherche/
         equipes/{@html}.en.html">
        <xsl:value-of select="$LeTypeProjet"/> <xsl:text> </xsl:text>
        <xsl:value-of select="projet"/>
     </fo:basic-link>
   </fo:block>
    <fo:block font-size= "25pt" text-align="center" space-before="1cm">
          <xsl:value-of select="projetdeveloppe"/>
    </fo:block>
   <fo:block font-size= "17.28pt" text-align="center" space-before="1cm">
     <xsl:for-each select ="UR/*">
        <fo:basic-link external-destination ="{@url}">
          <xsl:value-of select="@nom"/>
          <xsl:if test="position()!=last()"> - </xsl:if>
        </fo:basic-link>
     </xsl:for-each>
    </fo:block>
   <fo:block font-size= "10pt" font-style="normal"
       font-family="Helvetica" text-align="center" space-before="1cm">
       <fo:RATHEME><xsl:value-of select="theme"/></fo:RATHEME>
```

```
    </fo:block>
</xsl:template>
```

A <moreinfo> is typeset in italics.

```
<xsl:template match="moreinfo">
  <fo:block font-style="italic"> <xsl:apply-templates/> </fo:block>
</xsl:template>
```

This is a priori useless.

```
<xsl:template match="anchor">
  <fo:inline>
     <xsl:attribute name="id"><xsl:value-of select="@id"/></xsl:attribute>
  </fo:inline>
</xsl:template>
```

Translation of <keywords>. We take all the <term> elements.

```
<xsl:template match="keywords">
 <fo:block space-after.optimum="4pt">
   <fo:inline>
     <xsl:attribute name="font-weight">bold</xsl:attribute>
     <xsl:value-of select="./@titre"/>
   </fo:inline>
   <xsl:for-each select="term">
      <xsl:apply-templates select="."/> <xsl:call-template name="separateur.objet"/>
      </xsl:for-each>
 </fo:block>
</xsl:template>
```

This adds an id.

```
<xsl:template name="addID">
   <xsl:attribute name="id">
     <xsl:call-template name="idLabel"/>
   </xsl:attribute>
</xsl:template>
```

```
<xsl:template match="TeX">
   <TeX/>
</xsl:template>
<xsl:template match="LaTeX">
   <LaTeX/>
</xsl:template>
```

```
<xsl:include href="raweb3-table.xsl"/>
<xsl:include href="raweb3-makecolspec.xsl"/>
```

This is the end of the file.

```
</xsl:stylesheet>
```

7.3 Computing column specifications

We consider here a file that starts like this. This uses extensions to XSLT. It is taken from the TEI distribution, see Copyright notice in the preceding chapter.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```
  xmlns:fotex="http://www.tug.org/fotex"
  xmlns:exsl="http://exslt.org/common"
  exclude-result-prefixes="saxon exsl"
  extension-element-prefixes="saxon exsl fotex"
  xmlns:saxon="http://icl.com/saxon"
  xmlns:fo="http://www.w3.org/1999/XSL/Format" version="1.0">
```

The computations are the following. For each cell in the table, we construct a variable, named `$stuff`, that contains the value of the cell (some templates are applied). We construct a `<cell>` element, that has a `<col>` attribute whose value is the position of the cell; hence this algorithm does not work in the case where the cell spans multiple columns. The content of the cell is its width (i.e., is the width of the cell to typeset); this is hard to compute (consider two cells, one with an image of size 10cm, another one with a caption for it). In order to make this guess more usable, we added 100[4]. All these cells are put in a variable `$tds`. Consider for instance the case where the cells are of width 1, 2, 3, and 4, there are two rows, the first two cells are in the first row. We compute the total, namely 10. The columns totals are 4 and 6, so that the width of the columns are 40% and 60%; these two quantities are computed by sorting the cells by column. For each cell that is the first (has no preceding sibling) we compute the sum of the following siblings.

```
<xsl:template name="calculateTableSpecs">
  <xsl:variable name="tds">
     <xsl:for-each select=".//cell">
        <xsl:variable name="stuff">
           <xsl:apply-templates/>
        </xsl:variable>
        <cell>
            <xsl:attribute name="col"><xsl:number/></xsl:attribute>
            <xsl:value-of select="string-length($stuff) + 100"/>
        </cell>
     </xsl:for-each>
  </xsl:variable>
  <xsl:variable name="total">
      <xsl:value-of select="sum(exsl:node-set($tds)/cell)"/>
  </xsl:variable>
  <xsl:for-each select="exsl:node-set($tds)/cell">
     <xsl:sort select="@col" data-type="number"/>
     <xsl:variable name="c" select="@col"/>
     <xsl:if test="not(preceding-sibling::cell[$c=@col])">
       <xsl:variable name="len">
          <xsl:value-of select="sum(following-sibling::cell[$c=@col]) + current()"/>
       </xsl:variable>
       <xsl:text>
       </xsl:text>
       <fo:table-column column-number="{@col}"
           fotex:column-align="L" column-width="{$len div $total * 100}%" />
     </xsl:if>
  </xsl:for-each>
  <xsl:text></xsl:text> <!-- this is a newline character -->
</xsl:template>


</xsl:stylesheet>
```

[4]Why not use a random number? there should be a parameter in Tralics that inhibits this template.

7.4 Converting cells

This is another file. It is taken from the TEI distribution, see Copyright notice in the preceding chapter.

```
<xsl:stylesheet
  xmlns:fotex="http://www.tug.org/fotex"
  xmlns:exsl="http://exslt.org/common"
  exclude-result-prefixes="exsl"
  extension-element-prefixes="exsl"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
  xmlns:fo="http://www.w3.org/1999/XSL/Format"  >
```

The translation of `<cell>` is `<fo:table-cell>`. If the cols or rows attributes are at leat two, we add corresponding attributes to the result. If the cell, or its parent has a role attribute that says it is a label, we change the font to bold. Other attribute can also be given, see below. .

```
<xsl:template match="cell">
  <fo:table-cell>
    <xsl:if test="@cols &gt; 1">
      <xsl:attribute name="number-columns-spanned">
        <xsl:value-of select="@cols"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="@rows &gt; 1">
      <xsl:attribute name="number-rows-spanned">
        <xsl:value-of select="@rows"/>
      </xsl:attribute>
    </xsl:if>
   <xsl:call-template name="cellProperties"/>
   <fo:block>
    <xsl:choose>
      <xsl:when test="@role='label' or parent::row[@role='label']">
        <xsl:attribute name="font-weight">bold</xsl:attribute>
      </xsl:when>
    </xsl:choose>
    <xsl:apply-templates/>
   </fo:block>
  </fo:table-cell>
</xsl:template>
```

These are the possible cell properties. If the role is 'hi', we set the background color to 'silver'. In the case where the parent table has rend='`frame`', we add a frame around the cell. Only the first row has a border-before, only the last cell in the row has a border-end. Otherwise, we look at the properties of the cell, or of the row.

```
<xsl:template name="cellProperties" >
  <xsl:if test="@role='hi'">
    <xsl:attribute name="background-color">silver</xsl:attribute>
  </xsl:if>

  <xsl:choose>
    <xsl:when test="ancestor::table[1][@rend='frame']">
      <xsl:if test="not(parent::row/preceding-sibling::row)">
        <xsl:attribute name="border-before-style">solid</xsl:attribute>
      </xsl:if>
```

```
      <xsl:attribute name="border-after-style">solid</xsl:attribute>
      <xsl:if test="not(following-sibling::cell)">
        <xsl:attribute name="border-end-style">solid</xsl:attribute>
      </xsl:if>
      <xsl:attribute name="border-start-style">solid</xsl:attribute>
    </xsl:when>
    <xsl:otherwise>
      <xsl:if test="@left-border='true'">
        <xsl:attribute name="border-start-style">solid</xsl:attribute>
      </xsl:if>
      <xsl:if test="@right-border='true'">
        <xsl:attribute name="border-end-style">solid</xsl:attribute>
      </xsl:if>
      <xsl:if test="ancestor::row/@top-border='true'">
        <xsl:attribute name="border-before-style">solid</xsl:attribute>
      </xsl:if>
      <xsl:if test="ancestor::row/@bottom-border='true'">
        <xsl:attribute name="border-after-style">solid</xsl:attribute>
      </xsl:if>
      <xsl:if test="ancestor::row/@bottom-border='true'">
        <xsl:attribute name="border-after-style">solid</xsl:attribute>
      </xsl:if>
      <xsl:if test="@bottom-border='true'">
        <xsl:attribute name="border-after-style">solid</xsl:attribute>
      </xsl:if>
      <xsl:if test="@top-border='true'">
        <xsl:attribute name="border-before-style">solid</xsl:attribute>
      </xsl:if>
    </xsl:otherwise>
  </xsl:choose>
```

If the table is not tight, we add some padding to the cell.

```
  <xsl:if test="not(ancestor::table/@rend='tight')">
    <xsl:attribute name="padding">
       <xsl:value-of select="$tableCellPadding"/>
    </xsl:attribute>
  </xsl:if>
```

In the case where the cell has an horizontal alignment property we use it. Otherwise, we look at some database, and try to extract that information. This is not used by the Raweb.

```
  <xsl:choose>
    <xsl:when test="@halign">
      <xsl:attribute name="text-align">
         <xsl:value-of select="@halign"/>
      </xsl:attribute>
    </xsl:when>
    <xsl:otherwise>
      <xsl:variable name="thiscol">
        <xsl:value-of select="position()"/>
      </xsl:variable>
      <xsl:variable name="tid">...</xsl:variable>
      <xsl:variable name="align">... </xsl:variable>
      <xsl:choose>
```

```
1507            <xsl:when test="$align='R'">
1508                  <xsl:attribute name="text-align">right</xsl:attribute>
1509            </xsl:when>
1510            <xsl:when test="$align='L'">
1511                  <xsl:attribute name="text-align">left</xsl:attribute>
1512            </xsl:when>
1513            <xsl:when test="$align='C'">
1514                  <xsl:attribute name="text-align">center</xsl:attribute>
1515            </xsl:when>
1516            <xsl:otherwise>
1517              <xsl:if test="not($align='')">
1518                 <xsl:attribute name="text-align">
1519                     <xsl:value-of select="$align"/>
1520                 </xsl:attribute>
1521              </xsl:if>
1522            </xsl:otherwise>
1523        </xsl:choose>
1524      </xsl:otherwise>
1525    </xsl:choose>
1526  </xsl:template>
```

This is the end of the file.

```
1527  </xsl:stylesheet>
```

7.5 Customisation

We describe here the file raweb3-param.xsl. This is an adaptation of a file provided by the TEI. We changed the order of items, indicated which parameters are used, or unused.

```
1528  <xsl:stylesheet
1529    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
1530    xmlns:fo="http://www.w3.org/1999/XSL/Format">
```

Vertical space before and after a list. The `$exampleMargin` is used as left and right margins for figure captions.

```
1531  <xsl:param name="listAbove-1">6pt</xsl:param>
1532  <xsl:param name="listBelow-1">6pt</xsl:param>
1533  <xsl:param name="listAbove-2">4pt</xsl:param>
1534  <xsl:param name="listBelow-2">4pt</xsl:param>
1535  <xsl:param name="listAbove-3">0pt</xsl:param>
1536  <xsl:param name="listBelow-3">0pt</xsl:param>
1537  <xsl:param name="listAbove-4">0pt</xsl:param>
1538  <xsl:param name="listBelow-4">0pt</xsl:param>
1539  <xsl:param name="exampleMargin">12pt</xsl:param>
```

Quantities use to define list markers. U+2219 is bullet operator (defined to be `\ensuremath {\bullet}` in ucharacters.sty) is en dash (`\textendash`), U+002A is asterisk and U+002B is plus sign.

```
1540  <xsl:param name="bulletOne">&#x2219;</xsl:param>
1541  <xsl:param name="bulletTwo">&#x2013;</xsl:param>
1542  <xsl:param name="bulletThree">&#x002A;</xsl:param>
1543  <xsl:param name="bulletFour">&#x002B;</xsl:param>
```

Page dimensions and margins. The default values are used, except for the right margin (we use the same value as for the left margin). We use a single column. There is a template named hookDefinepagemasters that does nothing. This could be redefined to do some action. Idem for blockStartHook.

```
<xsl:param name="pageWidth">210mm</xsl:param>
<xsl:param name="pageHeight">297mm</xsl:param>
<xsl:param name="regionAfterExtent">12pt</xsl:param>
<xsl:param name="regionBeforeExtent">12pt</xsl:param>
<xsl:param name="bodyMarginBottom">24pt</xsl:param>
<xsl:param name="bodyMarginTop">24pt</xsl:param>
<xsl:param name="pageMarginTop">75pt</xsl:param>
<xsl:param name="pageMarginBottom">100pt</xsl:param>
<xsl:param name="pageMarginLeft">80pt</xsl:param>
<xsl:param name="pageMarginRight">150pt</xsl:param>
<xsl:param name="columnCount">1</xsl:param>
<xsl:template name="hookDefinepagemasters"/>
<xsl:template name="blockStartHook"/>
```

Fonts. The body and the headers use Times as font family. The two other fonts are used for typesetting <code> and <ident>, bu these two elements are not created by Tralics.

```
<xsl:param name="bodyFont">Times Roman</xsl:param>
<xsl:param name="divFont">Times Roman</xsl:param>
<xsl:param name="typewriterFont">Computer-Modern-Typewriter</xsl:param>
<xsl:param name="sansFont">Helvetica</xsl:param>
```

Font sizes. The main font size is 10pt, the quantity `$footnoteSize` is used for footnotes (8pt) and `$footnotenumSize` for footnote markers (is 7pt not too small?). We define a negative indentation for the section titles: `$headingOutdent`, but a positive paragraph indentation. Note: the first paragraph in a division is not indented.

```
<xsl:param name="bodyMaster">10</xsl:param>
<xsl:param name="bodySize">
 <xsl:value-of select="$bodyMaster"/><xsl:text>pt</xsl:text>
</xsl:param>
<xsl:param name="footnoteSize">8pt</xsl:param>
<xsl:param name="footnotenumSize">7pt</xsl:param>
<xsl:param name="headingOutdent">-3em</xsl:param>
<xsl:param name="parIndent">1em</xsl:param>
<xsl:param name="parSkip">0pt</xsl:param>
<xsl:param name="parSkipmax">12pt</xsl:param>
```

The following is used for a section title. We use a bold 18pt font. There is some space before and after: 12pt and 6pt.

```
<xsl:template name="setupDiv0">
 <xsl:attribute name="font-size">18pt</xsl:attribute>
 <xsl:attribute name="text-align">left</xsl:attribute>
 <xsl:attribute name="font-weight">bold</xsl:attribute>
 <xsl:attribute name="space-after">6pt</xsl:attribute>
 <xsl:attribute name="space-before.optimum">12pt</xsl:attribute>
 <xsl:attribute name="text-indent"><xsl:value-of select="$headingOutdent"/></xsl:attribute>
</xsl:template>
```

The following is used for a subsection (module) title. We use a bold 14pt font. There is some space before and after, but less than for a section: 9pt and 3pt.

```
<xsl:template name="setupDiv1">
 <xsl:attribute name="font-size">14pt</xsl:attribute>
 <xsl:attribute name="text-align">left</xsl:attribute>
 <xsl:attribute name="font-weight">bold</xsl:attribute>
 <xsl:attribute name="space-after">3pt</xsl:attribute>
 <xsl:attribute name="space-before.optimum">9pt</xsl:attribute>
 <xsl:attribute name="text-indent"><xsl:value-of select="$headingOutdent"/></xsl:attribute>
</xsl:template>
```

The following is used for a subsubsection title. We use a bold 12pt font. There is some space before and after, but less than for a subsection: 4pt and 2pt.

```
<xsl:template name="setupDiv2">
 <xsl:attribute name="font-size">12pt</xsl:attribute>
 <xsl:attribute name="text-align">left</xsl:attribute>
 <xsl:attribute name="font-weight">bold</xsl:attribute>
 <xsl:attribute name="font-style">italic</xsl:attribute>
 <xsl:attribute name="space-after">2pt</xsl:attribute>
 <xsl:attribute name="space-before.optimum">4pt</xsl:attribute>
 <xsl:attribute name="text-indent"><xsl:value-of select="$headingOutdent"/></xsl:attribute>
</xsl:template>
```

The following is used for a paragraph title. We use an italic 10pt font.

```
<xsl:template name="setupDiv3">
 <xsl:attribute name="font-size">10pt</xsl:attribute>
 <xsl:attribute name="text-align">left</xsl:attribute>
 <xsl:attribute name="font-style">italic</xsl:attribute>
 <xsl:attribute name="space-after">0pt</xsl:attribute>
 <xsl:attribute name="space-before.optimum">4pt</xsl:attribute>
 <xsl:attribute name="text-indent"><xsl:value-of select="$headingOutdent"/></xsl:attribute>
</xsl:template>
```

The following is used for a subparagraph title. We use a 10pt normal font.

```
<xsl:template name="setupDiv4">
 <xsl:attribute name="font-size">10pt</xsl:attribute>
 <xsl:attribute name="space-before.optimum">4pt</xsl:attribute>
 <xsl:attribute name="text-indent"><xsl:value-of select="$headingOutdent"/></xsl:attribute>
</xsl:template>
```

This contructs the caption of a figure. It is centered, with some indentation on both sides.

```
<xsl:template name="figureCaptionstyle">
  <xsl:attribute name="text-align">center</xsl:attribute>
  <xsl:attribute name="font-style">italic</xsl:attribute>
  <xsl:attribute name="end-indent">
   <xsl:value-of select="$exampleMargin"/>
  </xsl:attribute>
  <xsl:attribute name="start-indent">
   <xsl:value-of select="$exampleMargin"/>
  </xsl:attribute>
</xsl:template>
```

The template here could be used for table captions. If we compare with figures above, we see that there is some vertical space before and after the caption.

```
<xsl:param name="tableCaptionAlign">center</xsl:param>
<xsl:param name="spaceAroundTable">8pt</xsl:param>
<xsl:param name="spaceBelowCaption">4pt</xsl:param>
<xsl:param name="tableAlign">center</xsl:param>
<xsl:param name="tableCellPadding">2pt</xsl:param>
```

```
1624
1625 <xsl:template name="tableCaptionstyle">
1626   <xsl:attribute name="text-align">center</xsl:attribute>
1627   <xsl:attribute name="font-style">italic</xsl:attribute>
1628   <xsl:attribute name="end-indent">
1629    <xsl:value-of select="$exampleMargin"/>
1630   </xsl:attribute>
1631   <xsl:attribute name="start-indent">
1632    <xsl:value-of select="$exampleMargin"/>
1633   </xsl:attribute>
1634         <xsl:attribute name="space-before">
1635           <xsl:value-of select="$spaceAroundTable"/>
1636         </xsl:attribute>
1637   <xsl:attribute name="space-after">
1638           <xsl:value-of select="$spaceBelowCaption"/>
1639   </xsl:attribute>
1640   <xsl:attribute name="keep-with-next">always</xsl:attribute>
1641 </xsl:template>
```

Parameters for the TOC. What is `$tocSize`? the font size of the title of the toc? we use 14pt, see above. The two variables `$headingNumberSuffix` and `$tocNumberSuffix` explain that we should put a period-plus-space after numbers in the toc and headings. The variable `$numberHeadings` says that we want something to appear in the TOC, and `$numberDepth` says that we want in fact everything (in the HTML version, the TOC contains only sections and subsections, for the Pdf, we accept everything. In general it fits on a double page.)

```
1642 <xsl:param name="tocSize">16pt</xsl:param>
1643 <xsl:param name="div0Tocindent">0in</xsl:param>
1644 <xsl:param name="div1Tocindent">0.25in</xsl:param>
1645 <xsl:param name="div2Tocindent">0.5in</xsl:param>
1646 <xsl:param name="div3Tocindent">0.75in</xsl:param>
1647 <xsl:param name="div4Tocindent">1in</xsl:param>
1648 <xsl:param name="headingNumberSuffix">. </xsl:param>
1649 <xsl:param name="tocNumberSuffix">. </xsl:param>
1650 <xsl:param name="numberHeadings">true</xsl:param>
1651 <xsl:param name="numberDepth">9</xsl:param>
```

Parameters for lists.

```
1652 <xsl:param name="listRightMargin">10pt</xsl:param>
1653 <xsl:param name="listNormalIndent">15pt</xsl:param>
1654 <xsl:param name="listLeftGlossIndent">7mm</xsl:param>
1655 <xsl:param name="listLeftGlossInnerIndent">4mm</xsl:param>
1656 <xsl:param name="listLeftIndent">5pt</xsl:param>
1657 <xsl:param name="listItemsep">4pt</xsl:param>
```

Four names, useful if we decide to switch from English to another language. We use only 'Figure' and 'Table'. Note the space at the end; there is no space for the other words.

```
1658 <xsl:param name="figureWord">Figure </xsl:param>
1659 <xsl:param name="tableWord">Table </xsl:param>
1660 <xsl:param name="appendixWords">Appendix</xsl:param>
1661 <xsl:param name="biblioWords">Bibliography</xsl:param>
```

Overwritten by the Raweb.

```
1662 <xsl:param name="linkColor">black</xsl:param>
1663 <xsl:param name="pdfBookmarks"></xsl:param>
```

Not used by the Raweb. The variable `$autoScaleFigures` is not used: scaling must be explicit. The variable `$graphicsPrefix` is not used, because we assume that images are in the current directory, and there is no need to add a prefix. The variable `$smallSize` is not not used (see page 191 and following how smaller and larger fonts are used). The quantities `$authorSize`, `$dateSize` and `$titleSize` could be used for the title page. In the case of the Raweb, the title page is created by the code on page 58, this defines fonts and size.

```
<xsl:param name="autoScaleFigures"></xsl:param>
<xsl:param name="graphicsPrefix"></xsl:param>
<xsl:param name="smallSize">
 <xsl:value-of select="$bodyMaster * 0.9"/><xsl:text>pt</xsl:text>
</xsl:param>
<xsl:param name="authorSize">14pt</xsl:param>
<xsl:param name="dateSize">14pt</xsl:param>
<xsl:param name="titleSize">16pt</xsl:param>
```

The variable `$hyphenate` is unused: we always hyphenate. The variable `$alignment` is unused: we always justify. The `$language` variable is unused.

```
<xsl:param name="hyphenate">true</xsl:param>
<xsl:param name="alignment">justify</xsl:param>
<xsl:param name="language">US</xsl:param>
```

This defines sizes for quantities that are not used here. The quantities `$runSize` and `$runFont` are for running heads. We use the default font.

```
<xsl:param name="runSize">9pt</xsl:param>
<xsl:param name="runFont">sans-serif</xsl:param>
```

These are unused.

```
<xsl:param name="flowMarginLeft"></xsl:param>
<xsl:param name="giColor">orange</xsl:param>
<xsl:param name="identColor">blue</xsl:param>
<xsl:param name="activeLinebreaks"></xsl:param>
<xsl:param name="activePagebreaks"></xsl:param>
<xsl:param name="tocFront">true</xsl:param>
<xsl:param name="tocBack">true</xsl:param>
<xsl:param name="tocStartPage">1</xsl:param>
<xsl:param name="exampleSize">
   <xsl:value-of select="$bodyMaster * 0.8"/><xsl:text>pt</xsl:text>
</xsl:param>
<xsl:param name="exampleBefore">4pt</xsl:param>
<xsl:param name="exampleAfter">4pt</xsl:param>
<xsl:param name="OUCS"></xsl:param>
<xsl:param name="titlePage"></xsl:param>
<xsl:param name="divRunningheads"></xsl:param>
<xsl:param name="forcePageMaster"></xsl:param>
<xsl:param name="twoSided">true</xsl:param>
<xsl:param name="frontMulticolumns"></xsl:param>
<xsl:param name="bodyMulticolumns"></xsl:param>
<xsl:param name="backMulticolumns"></xsl:param>
<xsl:param name="sectionHeaders">true</xsl:param>
<xsl:param name="formatFrontpage">i</xsl:param>
<xsl:param name="formatBodypage">1</xsl:param>
<xsl:param name="formatBackpage">1</xsl:param>
<xsl:param name="formatAppendix">A.1.</xsl:param>
<xsl:param name="numberBackHeadings">A.1</xsl:param>
<xsl:param name="numberFrontHeadings">1</xsl:param>
```

```
<xsl:param name="captionInlinefigures"></xsl:param>
<xsl:param name="xrefShowTitle"></xsl:param>
<xsl:param name="xrefShowHead"></xsl:param>
<xsl:param name="xrefShowPage"></xsl:param>
<xsl:param name="minimalCrossRef"/>
<xsl:param name="numberHeadingsDepth">9</xsl:param>
<xsl:param name="prenumberedHeadings"></xsl:param>
<xsl:param name="biblSize">16pt</xsl:param>
<xsl:param name="indentBibl">1em</xsl:param>
<xsl:param name="spaceBeforeBibl">4pt</xsl:param>
<xsl:param name="spaceAfterBibl">0pt</xsl:param>
<xsl:param name="inlineTables"></xsl:param>
<xsl:param name="rowAlign">left</xsl:param>
<xsl:param name="makeTableCaption">true</xsl:param>
<xsl:param name="readColSpecFile"></xsl:param>
```

This is the end of the file.

```
</xsl:stylesheet>
```

Chapter 8

The DTDs

8.1 The Raweb DTD

In this chapter we give the Raweb DTD (old and new one). The file starts like this.

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

There is the Copyright notice of the file:

```
<!--
  Copyright 2002-2004, 2005 Jose Grimm/ INRIA/ Apics Project
  You have the right to use this file in any way, with the following
  restriction: you cannot distribute modifications of this file under
  its name raweb3.dtd, the name raweb.dtd,
  nor rawebXXXX.dtd,  where XXXX are four digits like 2004.
  The name 'raweb3.dtd' is associated to the raweb DTD of YEAR 2003.
-->
```

We define here the `&` and `<` entities. I'm not sure that these definitions are required.

```
<!ENTITY amp "&#x26;#x26;" >
<!ENTITY lt  "&#x26;#x3C;" >
```

We define here some entities. They are used to typeset things like 1^{st}, 2^{nd}, etc., in French.

```
<!ENTITY  ier "<hi rend='sup'>er</hi>">
<!ENTITY  iers "<hi rend='sup'>ers</hi>">
<!ENTITY  iere "<hi rend='sup'>re</hi>">
<!ENTITY  ieres "<hi rend='sup'>res</hi>">
<!ENTITY  ieme "<hi rend='sup'>e</hi>">
<!ENTITY  iemes "<hi rend='sup'>es</hi>">
<!ENTITY  numero "n<hi rend='sup'>o</hi>">
<!ENTITY  Numero "N<hi rend='sup'>o</hi>">
```

This includes the MathML DTD. Note that `<ident>` and `<list>` are defined here and in the MathML DTD, they are in a different namespace.

```
<!ENTITY % list.qname "m:list" >
<!ENTITY % ident.qname "m:ident" >

<!ENTITY % mathml PUBLIC "mathml" "mathml2.dtd">
%mathml;
```

8.1.1 General purpose elements

We list here the elements that can appear in the body of a division. There was a <cit> element here, it should appear only inside a <p>. Since version 2.5, Tralics inserts a \leavevmode before a \cite. Note that <formula>, <table> or <figure> can be inline or not; inline versions are refused here.

```
<!ENTITY % tei-aux "(p | list | note | formula | table | figure)+" >
```

We list here elements that can only appear in certain places (for instance in a title). In the case of a <formula>, it should be inline.

```
<!ENTITY % texte-restreint
  "ident | code |  hi | term | ref | xref | formula | TeX | LaTeX" >
```

This is the concatenation of the two lists, with <cit> added.

```
<!ENTITY % texte-general
   "ident | code | hi | term| ref| xref | formula | cit | label | list |
     note | figure | table | TeX | LaTeX" >
```

The TEI defines a lot of common attributes. Here we use only two of them. Note: the id is used only in the following cases: a division (including modules, and sections like 'fondements'), an item in a list, a footnote, a table, a figure, a formula, an entry in the bibliography.

```
<!ENTITY % tei-common-atts
        'id ID #IMPLIED rend CDATA #IMPLIED'>
```

Definition of <code *common-atts*>*text*</code>. The content of the element is assumed to be some verbatim material, hence it contains only text. Tralics does not produce directly such an element.

```
<!ELEMENT code (#PCDATA) >
<!ATTLIST code  %tei-common-atts; >
```

Definition of <ident *common-atts*>*text*</ident>. The content of the element is assumed to be the name of an identifier, hence it contains only text. Tralics does not produce directly such an element.

```
<!ELEMENT ident (#PCDATA) >
<!ATTLIST ident %tei-common-atts; >
```

Definition of <cit rend=*A*><ref></cit>. The content of the element is a reference to the bibliography; Tralics sets the rend attribute to 'foot' in the case of \footcite.

```
<!ELEMENT cit (ref) >
<!ATTLIST cit rend CDATA #IMPLIED >
```

Definition of <list type=*X*> *body*</list>. This was simplified a lot, because we removed all references to <anchor>. As a result, a list can have an optional title (a <head>, but Tralics does not produce it), followed by a sequence of <item>. If the list has type gloss or description, the content is a sequence of <label> plus <item>.

```
<!ELEMENT list
        ( head?, (item* | (label, item)+)) >
<!ATTLIST list
        %tei-common-atts;
        type (simple|gloss|ordered|description)  "simple" >
```

Definition of <item id=*A*> *text*</item>. The content can be characters, paragraphs, and some general text (Tralics produces <p> elements in general). You can make a reference to an item; the rend attribute is not used.

```
<!ELEMENT item
        (#PCDATA | %texte-general; | p )* >
```

```
<!ATTLIST item
        id ID #IMPLIED
        rend CDATA #IMPLIED >
```

Definition of `<label> text </label>`. The content is the label of an item; it is the optional argument of the `\item` command. Only simple text is accepted (in particular, if you want a bullet, you should use `\textbullet` rather than `\bullet`).

```
<!ELEMENT label
        (#PCDATA | %texte-restreint; )* >
<!ATTLIST label %tei-common-atts; >
```

This defines the content of a division; at level 2, we accept only divisions of level 3 or 4. We need two variables for each level, since we cannot give a name `foo` to `(div3|div4)` and use it like `foo+` or `foo*`.

```
<!ENTITY % div0-textp "(div1|div2|div3|div4)+">
<!ENTITY % div0-texts "(div1|div2|div3|div4)*">
<!ENTITY % div1-textp "(div2|div3|div4)+">
<!ENTITY % div1-texts "(div2|div3|div4)*">
<!ENTITY % div2-textp "(div3|div4)+">
<!ENTITY % div2-texts "(div3|div4)*">
<!ENTITY % div3-textp "(div4)+">
<!ENTITY % div3-texts "(div4)*">
```

We explain here what can be put in the header of a module: participants, keywords, or moreinfo.

```
<!ENTITY % particip "participant|participants|participante|participantes" >
<!ENTITY % ramodule-header "(moreinfo|keywords|%particip;)*">
```

This describes the attributes for a division. For the Raweb, only id is used, rend and type are ignored.

```
<!ENTITY % tei-div-atts '
        %tei-common-atts;
        type CDATA #IMPLIED '>
```

This describes a `<module>`. There are some attributes: the html attribute is marked 'required', but it is nowadays unused, and id is preferred. The topic attribute, if present, must be a reference to a `<topic>` element (identified by its num attribute). Other attributes are unused. The content is: a `<head>` that indicates the title of the module, followed by some meta data (participants, keywords, more info), and some text (paragraphs, formulas, etc.) or divisions (`<div2>`, `<div3>`, etc.).

```
<!ELEMENT module
    (head, %ramodule-header;, (%div1-textp; | (%tei-aux;, %div1-texts;))) >

<!ATTLIST module %tei-div-atts;
       html CDATA #REQUIRED
       topic CDATA #IMPLIED >
```

A `<div2>` is like a `<module>`; but there are no topic and no html attribute. It is one level below a module.

```
<!ELEMENT div2
    (head, %ramodule-header;, (%div2-textp; | (%tei-aux;, %div2-texts;))) >
<!ATTLIST div2 %tei-div-atts; >
```

A `<div3>` is like a `<div2>`. It is two levels below a module.

```
<!ELEMENT div3
   (head, %ramodule-header;, (%div3-textp; | (%tei-aux;, %div3-texts;))) >
<!ATTLIST div3 %tei-div-atts; >
```

A <div4> is like a <div2>. It is three levels below a module. The Raweb does not define <div5>!

```
<!ELEMENT div4
        (head, %ramodule-header;, %tei-aux;) >
<!ATTLIST div4 %tei-div-atts; >
```

Definition of a <table>. It is formed of a <head>, the caption of the table, optional, and a sequence of <row>. The rows and cols attributes are defined by the TEI as the number of rows in the table and the number of columns per row. They are not used by the Raweb.

```
<!ELEMENT table
       (head*, row+) >

<!ATTLIST table
        %tei-common-atts;
        rows NMTOKEN #IMPLIED
        cols NMTOKEN #IMPLIED >
```

A <row> is a sequence of <cell>. In the initial versions, a row could contain other things, like a table, but this seems stupid. The attributes top-border and bottom-border can be set to true if you want a border for each cell in the row. The space-before controls the space between this row and the preceding one. The role could be used to emphasize (for instance change the background color, for the first row).

```
<!ELEMENT row
        (cell)+ >
<!ATTLIST row
        %tei-common-atts;
        top-border (true|false) "false"
        bottom-border (true|false) "false"
        space-before CDATA #IMPLIED
        role CDATA "data" >
```

A <cell> can contain text, or more complicated things. It can be empty. Normally, the horizontal or vertical span is one; this can be changed by setting the attributes rows and cols. The attributes right-border, left-border, top-border, and bottom-border can be set to true if you want a border on either side. The halign indicates horizontal alignment (it can be left, center, or right). The role could be used to emphasize (for instance change the background color, for the first column).

```
<!ELEMENT cell
        (#PCDATA | %texte-general;)* >
<!ATTLIST cell
        %tei-common-atts;
        role CDATA "data"
        rows NMTOKEN "1"
        cols NMTOKEN "1"
        right-border (true|false) "false"
        left-border (true|false) "false"
        halign CDATA #IMPLIED
        top-border (true|false) "false"
        bottom-border (true|false) "false"
   >
```

A <figure> element has a rend attribute. If this is 'inline', the content should be empty, and the file attribute should define an image. You can also specify a non-inline figure, with no attributes and a content formed uniquely of a caption (a <head> element), or you could specify an optional caption and a sequence of paragraphs (no file in this case). In the case a file attribute is given, this should be the name of an image; in this case, you may say framed='true' if you want a frame around the image; you can set width and height if you want to alter the dimensions of the image, or scale, if you want to rescale it (you cannot give scale together with height or depth); you can specify angle if you want to turn the image.

```
<!ELEMENT figure
        (head?, p*)>
<!ATTLIST figure
        id ID #IMPLIED
        rend (inline|float) "float"
        file CDATA #IMPLIED
        framed CDATA #IMPLIED
        width CDATA #IMPLIED
        height CDATA #IMPLIED
        scale CDATA #IMPLIED
        angle CDATA #IMPLIED>
```

You can say <simplemath>N</simplemath> if you want the TeX equivalent of N. This must be in a formula; it could be rendered as \textit{N}. It contains a single character.

```
<!ELEMENT simplemath (#PCDATA) >
```

A <formula> is a wrapper for a math expression; this could be a simple math expression (as above), or a true math expression (as defined by MathML). It has an attribute type. Normally, simple math expressions should be inline. You can reference a formula via its id, but this works only for display math formulas.

```
<!ELEMENT formula
        (simplemath |math) >
<!ATTLIST formula
        %tei-common-atts;
        type (inline|display) "inline" >
```

A <keywords> element contains a list of <term>. The title can be used to typeset the list.

```
<!ELEMENT keywords (term+) >
<!ATTLIST keywords %tei-common-atts;
        titre CDATA #FIXED "Key words: "    >
```

A <term> is a keyword in a list. Attributes are currently unused.

```
<!ELEMENT term
        (#PCDATA | %texte-restreint;)* >
<!ATTLIST term
        %tei-common-atts;
        type CDATA #IMPLIED >
```

A <p> element is a paragraph. It contains some text (inline math, citations, etc). It is generally indented (but the first paragraph is a section, could have zero indentation), unless noindent is true. The attribute spacebefore can be used if more vertical space is wanted before the start of the paragraph.

```
<!ELEMENT p
        (#PCDATA | %texte-general; )* >

<!ATTLIST p
```

```
137         %tei-common-atts;
138         spacebefore CDATA #IMPLIED
139         noindent CDATA #IMPLIED>
```

The <hi> element can be used to hilight some text; the rend attribute explains how; on page 132 we list the different values accepted in HTML, and on page 191, we give a longer list for the Pdf case.

```
140 <!ELEMENT hi
141         (#PCDATA | %texte-general; )* >
142 <!ATTLIST hi
143         id ID #IMPLIED
144         rend CDATA #REQUIRED >
```

The <ref> element defines a reference to some element whose id is the value of the target attribute in the current document; its content is in general empty. The TEI says that the type is IDREFS, and the <ref> element can point to zero, one and more objects; the Raweb says that one and only one target must be given. The content of the element is useful only if the reference points to the bibliography (i.e., is in a <cit>). The type attribute is not used by the Raweb.

```
145 <!ELEMENT ref
146         (#PCDATA | %texte-general; )* >
147 <!ATTLIST ref
148         %tei-common-atts;
149         type CDATA #IMPLIED
150         target IDREF #IMPLIED >
```

The <xref> element defines a reference to an external document, defined by the url attribute (which should be a valid URL). Its content could be the value of the link, or maybe something else; it defines the zone in which the link is active. The type attribute is not used by the Raweb.

```
151 <!ELEMENT xref
152         (#PCDATA | %texte-general; )* >
153 <!ATTLIST xref
154         %tei-common-atts;
155         type CDATA #IMPLIED
156         url CDATA #IMPLIED>
```

The <head> element can be used as a section title or a caption. Attributes are ignored.

```
157 <!ELEMENT head
158         (#PCDATA | %texte-general; )* >
159 <!ATTLIST head
160         %tei-common-atts;
161         type CDATA #IMPLIED >
```

A <note> element can be used for footnotes, for instance if you say place='foot'. Other attributes like rend, anchored and target are currently ignored. Paragraphs are allowed.

```
162 <!ELEMENT note
163         (#PCDATA | %texte-general; |  p )* >
164 <!ATTLIST note
165         id ID #IMPLIED
166         rend CDATA #IMPLIED
167         type CDATA #IMPLIED
168         place CDATA "unspecified"
169         anchored (yes | no) "yes"
170         target IDREFS #IMPLIED >
```

The <anchor/> element is currently unused. We do not show the attribute list.

```
<!ELEMENT anchor EMPTY >
```

8.1.2 Elements specific to the Raweb

The main element is <raweb>; It is formed by a <accueil>, an optional <moreinfo>, and ten sections. All sections should be present, the mandatory ones are <composition> (who is in the team), <presentation> (a short presentation of the team), <resultats> (new result of the teams this year) and the bibliography. The language attribute indicates the language of the document (currently English), the year attribute indicates the year, and the creator holds the name of the software that created the document.

```
<!ELEMENT raweb (accueil, moreinfo?, composition, presentation,
  fondements?,domaine?, logiciels?, resultats,contrats?,international?,
  diffusion?,biblio) >
<!ATTLIST raweb year CDATA #IMPLIED >
<!ATTLIST raweb language CDATA #IMPLIED >
<!ATTLIST raweb creator CDATA #IMPLIED >
```

The <composition> element is one of the ten sections of the Raweb, as such it has some constant attributes like titre, html and numero. It has a possible id (making a reference to a section is not a good idea, because no HTML page is associated to a section; in the case without topics, a section is a continuous sequence of pages, so a link to the first page could be used; this is no more true in the case with topics). This section contain an optional <moreinfo> and a sequence of <catperso> elements.

```
<!ELEMENT composition (moreinfo?,catperso+)>
<!ATTLIST composition
           titre CDATA #FIXED "Team"
           html CDATA #FIXED "composition"
           numero CDATA #FIXED "1"
           id ID #IMPLIED>
```

The <presentation> section contains some <module> that explain briefly the main objectives of the Team.

```
<!ELEMENT presentation (module+) >
<!ATTLIST presentation
          titre CDATA #FIXED "Overall Objectives"
          numero CDATA #FIXED "2"
          id ID #IMPLIED>
```

The <fondements> section contains some <module> that explain the scientific foundations of the research of the Team.

```
<!ELEMENT fondements (module+) >
<!ATTLIST fondements
          titre CDATA #FIXED "Scientific Foundations"
          numero CDATA #FIXED "3"
          id ID #IMPLIED>
```

The <domaine> section contains some <module> that explain the application domains of the research of the Team.

```
<!ELEMENT domaine (module+) >
<!ATTLIST domaine
        titre CDATA #FIXED "Application Domains"
        numero CDATA #FIXED "4"
        id ID #IMPLIED>
```

The <logiciels> section contains some <module> that describe the software developed in the Team.

```
<!ELEMENT logiciels (module+) >
<!ATTLIST logiciels
        titre CDATA #FIXED "Software"
        numero CDATA #FIXED "5"
        id ID #IMPLIED>
```

The <resultats> section is the main section of the Raweb; it contains some <module> that explain the results of the Team for the current year.

```
<!ELEMENT resultats (module+) >
<!ATTLIST resultats
        titre CDATA #FIXED "New Results"
        numero CDATA #FIXED "6"
        id ID #IMPLIED>
```

The <contrats> section contains some <module> that explain the how the team is funded (it lists the contracts with industry, etc.)

```
<!ELEMENT contrats (module+) >
<!ATTLIST contrats
        titre CDATA #FIXED "Contracts and Grants with Industry"
        numero CDATA #FIXED "7"
        id ID #IMPLIED>
```

The <international> section contains some <module> that explain national, european, and international activities of the Team.

```
<!ELEMENT international (module+) >
<!ATTLIST international
        titre CDATA #FIXED "Other Grants and Activities"
        numero CDATA #FIXED "8"
        id ID #IMPLIED>
```

The <diffusion> section contains some <module> that explain the how the research of the Team is diffused (teaching, etc.)

```
<!ELEMENT diffusion (module+) >
<!ATTLIST diffusion
        titre CDATA #FIXED "Dissemination"
        numero CDATA #FIXED "9"
        id ID #IMPLIED>
```

The <accueil> element provides some information about the Team. The <addresse> element is obsolete, replaced by <UR>. The <typeprojet> is obsolete, replaced by the isproject attribute. The html attribute defines the name of the Team, using only ASCII characters, like 'Miro'; the <projet> element can be more complicated, like 'Miró', and the <projetdeveloppe> may contain "Systèmes à objets, types et prototypes : sémantique et validation" (whenever possible, the language should be the same as in the main document).

```
<!ELEMENT accueil (theme,projet,projetdeveloppe,UR,topic*) >
<!ATTLIST accueil html CDATA #REQUIRED >
<!ATTLIST accueil isproject (true|false) "true">
<!ELEMENT typeprojet (#PCDATA)>
<!ELEMENT addresse (#PCDATA) >
```

The <theme> should be one of com, cog, num, bio, sym.

```
<!ELEMENT theme (#PCDATA)>
<!ELEMENT projet (#PCDATA|hi)*>
<!ELEMENT projetdeveloppe (#PCDATA|hi)* >
```

The <UR> element contains some (at least one) references to Inria Research Units. Currently, there are six of them.

```
<!ELEMENT UR (URSophia|URRocquencourt|URRhoneAlpes|URRennes|URLorraine|URFuturs)+>
```

The definition of <URRocquencourt/> includes the full name (Rocquencourt) and a link to the official web page.

```
<!ELEMENT URRocquencourt EMPTY>
<!ATTLIST URRocquencourt
 url  CDATA #FIXED "http://www.inria.fr/inria/organigramme/fiche_ur-rocq.en.html"
 nom  CDATA #FIXED "Rocquencourt" >
```

Same for <URRennes>.

```
<!ELEMENT URRennes EMPTY>
<!ATTLIST URRennes
 url CDATA #FIXED "http://www.inria.fr/inria/organigramme/fiche_ur-ren.en.html"
 nom CDATA #FIXED "Rennes" >
```

Same for <URSophia>. Is the official name "Sophia-Antipolis" with a dash or without? the official web site[1] uses no dash, on Inria's front page (http://www-sop.inria.fr/) there are sometimes dashes, but not always.

```
<!ELEMENT URSophia EMPTY>
<!ATTLIST URSophia
 url CDATA #FIXED "http://www.inria.fr/inria/organigramme/fiche_ur-sop.en.html"
 nom CDATA #FIXED "Sophia Antipolis" >
```

Some for <URLorraine>.

```
<!ELEMENT URLorraine EMPTY>
<!ATTLIST URLorraine
 url CDATA #FIXED "http://www.inria.fr/inria/organigramme/fiche_ur-lor.en.html"
 nom CDATA #FIXED "Lorraine" >
```

Same for <URRhoneAlpes>.

```
<!ELEMENT URRhoneAlpes EMPTY>
<!ATTLIST URRhoneAlpes
 url CDATA #FIXED "http://www.inria.fr/inria/organigramme/fiche_ur-ra.en.html"
 nom CDATA #FIXED "Rhône-Alpes" >
```

Same for <URFuturs>.

```
<!ELEMENT URFuturs EMPTY>
<!ATTLIST URFuturs
 url CDATA #FIXED "http://www.inria.fr/inria/organigramme/fiche_ur-futurs.en.html"
 nom CDATA #FIXED "Futurs" >
```

Definitions of four similar elements. They contain a list of <pers>. You should use an 's' if more than one person is concerned. Use an 'e' if only woman are concerned. This distinction is useless in English.

```
<!ELEMENT participants (pers)+ >
<!ELEMENT participantes (pers)+ >
<!ELEMENT participante (pers)+ >
<!ELEMENT participant (pers)+ >
```

[1] http://www.sophia-antipolis.org/

```
<!ATTLIST participants titre CDATA #FIXED "Participants: ">
<!ATTLIST participantes titre CDATA #FIXED "Participants: ">
<!ATTLIST participante titre CDATA #FIXED "Participant: ">
<!ATTLIST participant titre CDATA #FIXED "Participant: ">
```

The <catperso> element defines a category of staff. It has a title (for instance "Boss"), and is followed by the list of all persons in the category (for instance, the big boss, the small boss, etc). Usually, alphabetic order is used.

```
<!ELEMENT catperso (head,pers+)>
```

A <pers> element has two attribute: first name and last name. It can have a content, that explains the position of the person. We allow footnotes, links, font changes, etc. Note that the whole element (including the name) should fit on a line.

```
<!ELEMENT pers (#PCDATA|hi|note|xref|ref)* >
<!ATTLIST pers prenom CDATA #REQUIRED
               nom CDATA #REQUIRED>
```

A <moreinfo> contains paragraphs. In can be used globally for the Raweb, or in modules, for adding a short sentence (like "Work done in collaboration with Team X").

```
<!ELEMENT moreinfo (p+) >
```

A <topic> contains a <t_titre>, that contains text, and has a unique attribute (currently a number...). The text describes the topic. Most modules can reference a topic.

```
<!ELEMENT topic (t_titre) >
<!ELEMENT t_titre (#PCDATA) >
<!ATTLIST topic num CDATA #IMPLIED>
```

The <biblio> element contains the whole bibliography, all <citation> elements. It has three constant attributes: the title, the section number, the HTML file name.

```
<!ELEMENT biblio (citation)* >
<!ATTLIST biblio
        html CDATA #FIXED "bibliography"
        titre CDATA #FIXED "Bibliography"
        numero CDATA #FIXED "10">
```

Definition of <TeX/> and <LaTeX/>. They are empty, and should be rendered as TEX and LATEX.

```
<!ELEMENT TeX EMPTY>
<!ELEMENT LaTeX EMPTY>
```

8.1.3 The bibliography

These are all the possible children of a citation. The <xref> element is defined above, others are specific to the bibliography.

```
<!ENTITY % bibliostuff "bnote|bauteurs|beditеur|btitle|borganization|
     bschool|byear|bmonth|xref|bseries|bnumber|bvolume|bedition|
     binstitution|baddress|bpages|bhowpublished|bbooktitle
     |bpublisher|bjournal|bchapter|btype|bdoi">
```

The <citation> contains the elements indicated above. It has five attributes. The type attribute reflects the BibTeX type. Entries that have a different type are lost by the Raweb mechanism. There are three required attributes <key>, <from>, <id>. The key is the quantity that will appear in the text, before the reference (however, a postprocessor could re-arrange the list, sort it, and recompute the key). The id identifies the reference, so that it can be referenced. The from

attribute is something required by the Raweb, it is used for sorting the citations. Finally, userid is optional: this is the cite key, that appears in the BibTeX source. Useful for debug.

```
<!ELEMENT citation (%bibliostuff;)*>
<!ATTLIST citation key CDATA #REQUIRED
                   userid CDATA #IMPLIED
                   id ID #REQUIRED
                   type (book|booklet|proceedings|phdthesis|article|inbook|
          incollection|inproceedings|conference|manual|techreport|coursenotes
         |unpublished |misc|masterthesis|mastersthesis) #REQUIRED
                   from (year|foot|refer) #REQUIRED >
```

Definition of <etal/>. There is a nom attribute, that helps typesetting this element. You can use this element in an author list, like 'etc.' in an enumeration.

```
<!ELEMENT etal EMPTY>
<!ATTLIST etal nom CDATA #FIXED "et al." >
```

Definition of <bpers prenom=*A* nom=*B* part=*C* junior=*D*/> In the case where the BibTeX entry is "de la Porte, Fils, {\'Emile}", the prenom would be '`Émile`', the nom would be '`Porte`', the part would be '`de la`' and the junior would be '`Fils`'.

```
<!ELEMENT bpers EMPTY>
<!ATTLIST bpers prenom CDATA #REQUIRED
                part CDATA #IMPLIED
                nom CDATA #REQUIRED
                junior CDATA #IMPLIED>
```

Definition of <bauteurs> and <bediteur>. This is the list of authors or editors of a citation. The content should be a sequence of <bpers>, optionally followed by a <etal>. The bname attribute is used for typesetting the element.

```
<!ELEMENT bauteurs (bpers|etal)* >
<!ATTLIST bauteurs bname CDATA #FIXED "authors" >
<!ELEMENT bediteur (bpers|etal)* >
<!ATTLIST bediteur bname CDATA #FIXED "editors" >
```

For the elements that follow, explanations are taken from the LaTeX companion [6]. The bname attribute can be used to typeset the field.

The organization that sponsors a conference or that publishes a manual.

```
<!ELEMENT borganization (#PCDATA) >
<!ATTLIST borganization bname CDATA #FIXED "organisation" >
```

Institution sponsoring a technical report. We allow font changes here.

```
<!ELEMENT binstitution (#PCDATA|hi)* >
<!ATTLIST binstitution bname CDATA #FIXED "institution" >
```

Usually the address of the publisher or other institution. For major publishing houses, just give the city. For small publishers, specifying the complete address might help the reader.

```
<!ELEMENT baddress (#PCDATA) >
<!ATTLIST baddress bname CDATA #FIXED "address" >
```

Journal name. Abbreviations are provided for many journals (but Tralics knows none of them). Font changes are allowed.

```
<!ELEMENT bjournal (#PCDATA|hi)* >
<!ATTLIST bjournal bname CDATA #FIXED "journal" >
```

The name of a series or set of books. When citing an entire book, the <btitle> field gives its title and an optional <series> field gives the name of a series or multivolume set in which the book is published.

```
<!ELEMENT bseries (#PCDATA|hi)* >
<!ATTLIST bseries bname CDATA #FIXED "series" >
```

Title of a book, part of which is being cited. For book entries, use the <btitle> field. This allows font changes.

```
<!ELEMENT bbooktitle (#PCDATA|hi)* >
<!ATTLIST bbooktitle bname CDATA #FIXED "booktitle" >
```

The publishers' name.

```
<!ELEMENT bpublisher (#PCDATA |hi)* >
<!ATTLIST bpublisher bname CDATA #FIXED "publisher" >
```

One or more page numbers or range of numbers (e.g. 42-111 or 7,41,,73-97 or 43+, where the '+' indicates pages that do not form a simple range).

```
<!ELEMENT bpages (#PCDATA) >
<!ATTLIST bpages bname CDATA #FIXED "pages" >
```

A chapter (or section or whatever) number.

```
<!ELEMENT bchapter (#PCDATA) >
<!ATTLIST bchapter bname CDATA #FIXED "chapter" >
```

The type of a technical report (e.g. "Research Notes"). This name is used instead of the default "Technical Report". For the type 'phdthesis' you could use the term "Ph.D. dissertation" by specifying type="{Ph.D.} dissertation". Similarly, for the <inbook> and <incollection> entry types you can get "section 1.2" instead of the default "chapter 1.2" with chapter = "1.2", type="Section". Note: the <btype> field is set by Tralics; the semantics is different. Currently, the bname value of <bchapter> is used instead of the quantity given here. This might change in a future version.

```
<!ELEMENT btype (#PCDATA|hi)* >
<!ATTLIST btype bname CDATA #FIXED "type" >
```

How something strange has been published. It can contain an external link.

```
<!ELEMENT bhowpublished (#PCDATA|xref|hi)* >
<!ATTLIST bhowpublished bname CDATA #FIXED "howpublished" >
```

The edition of a book (e.g. "Second"). This should be an ordinal, and should have the first letter capitalized, as shown above; the standard styles convert to lowercase when necessary. Note. The Raweb leaves this currently unchanged. Thus, be careful.

```
<!ELEMENT bedition (#PCDATA) >
<!ATTLIST bedition bname CDATA #FIXED "edition" >
```

The number of a journal, magazine, technical report, or work in a series. An issue of a journal or magazine is usually identified by its volume and number; a technical report normally has a number; and sometimes books in a named series carry numbers.

```
<!ELEMENT bnumber (#PCDATA) >
<!ATTLIST bnumber bname CDATA #FIXED "number" >
```

The volume of a journal or multivolume book.

```
<!ELEMENT bvolume (#PCDATA|hi)* >
<!ATTLIST bvolume bname CDATA #FIXED "volume" >
```

The month in which this work was published or, for an unpublished work, in which it was written. For reasons of consistency, the standard three-letter abbreviations (jan, feb, mar, etc.)

should be used. Note. Currently, Tralics replaces abbreviations by values; maybe we could add an attribute that says: this is an abbreviation.

```
<!ELEMENT bmonth (#PCDATA) >
<!ATTLIST bmonth bname CDATA #FIXED "month" >
```

The year of publication or, for an unpublished work, the year it was written. Generally, it should consist of four numerals, such as `1984`, although the standard styles can handle any year whose last four nonpunctuation characters are numerals, such as `about 1984`.

```
<!ELEMENT byear (#PCDATA|hi)* >
<!ATTLIST byear bname CDATA #FIXED "year" >
```

This is the doi (Digital Object Identifier). Use it whenever possible.

```
<!ELEMENT bdoi (#PCDATA)* >
<!ATTLIST bdoi bname CDATA #FIXED "DOI" >
```

Any additional information that can help the reader. This can contain math formulas. It can contain an external link. It can contain a link to the bibliography.

```
<!ELEMENT bnote (#PCDATA|xref|hi|cit|formula)* >
<!ATTLIST bnote bname CDATA #FIXED "note" >
```

The name of the school where the thesis was written.

```
<!ELEMENT bschool (#PCDATA|hi)* >
<!ATTLIST bschool bname CDATA #FIXED "school" >
```

The work's title, typed as explained in Section 13.2.2. Note: the companion explains how BibTeX converts some uppercase letters into lower case ones. This does not apply here.

```
<!ELEMENT btitle (#PCDATA|hi|TeX|LaTeX|formula)* >
<!ATTLIST btitle bname CDATA #FIXED "title" >
```

8.1.4 Research Reports

This DTD could also be used for other documents, like a research report. We declare here divisions of level zero and one.

```
<!ELEMENT div0
    (head, (%div0-textp; | (%tei-aux;, %div0-texts;))) >
<!ELEMENT div1
    (head, (%div1-textp; | (%tei-aux;, %div1-texts;))) >
```

The title page of a Research Report will contain the following items.

```
<!ELEMENT RRstart (UR,title, etitle, projet, theme, motcle, keyword,
   resume, abstract, author,date, RRnumber)*>
```

The `<UR>` element is described above, it indicates in which Research Unit is located the author (since there can be more than one author, more than one UR could be given, but this is refused by LaTeX). The quantities `<projet>` and `<theme>` define the name of the Team and its theme (this can be repeated). We have then `<title>` and `<etitle>`, the title of the report, in French and English, we have `<motcle>` and `<keyword>`, the keywords in French and English. The content could be more elaborate: for instance we could allow math in the title or keywords. We have also `<resume>` and `<abstract>`, the French and English abstract (paragraphs are allowed here; the abstract should be less than half of a page in length). The element `<auth>` defines a single author, while `<author>` defines a list of authors.

```
<!ELEMENT title (#PCDATA|hi|LaTeX)* >
<!ELEMENT etitle (#PCDATA|hi|LaTeX)* >
<!ELEMENT resume (#PCDATA|hi|p|LaTeX)* >
```

```
<!ELEMENT abstract (#PCDATA|hi|p|LaTeX)* >
<!ELEMENT motcle (#PCDATA|hi|LaTeX)* >
<!ELEMENT keyword (#PCDATA|hi|LaTeX)* >
<!ELEMENT RRnumber (#PCDATA)* >
<!ELEMENT date (#PCDATA)* >
<!ELEMENT author (auth)* >
<!ELEMENT auth (#PCDATA)* >
```

The could be the document element for a research report.

```
<!ELEMENT rr (RRstart,div0*)>
<!ATTLIST rr language CDATA #IMPLIED type CDATA #IMPLIED>
```

8.2 The raweb2 DTD

This is the start of the file. Current version is 1.1.2.3, dated 2004/12/16.

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

For some reason, every element has a prefix; this is currently empty.

```
<!ENTITY % prefix "">
```

A name is given for the Xlink namespace.

```
<!ENTITY % XLINK.xmlns "http://www.w3.org/1999/xlink" >
```

The DTD starts by listing everything in alphabetic order.

```
<!ENTITY % anchor        "%prefix;anchor">
<!ENTITY % b             "%prefix;b">
<!ENTITY % big           "%prefix;big">
<!ENTITY % caption       "%prefix;caption">
<!ENTITY % code          "%prefix;code">
<!ENTITY % descriptionlist "%prefix;descriptionlist">
<!ENTITY % em            "%prefix;em">
<!ENTITY % LaTeX         "%prefix;LaTeX">
<!ENTITY % TeX           "%prefix;TeX">
<!ENTITY % object        "%prefix;object">
<!ENTITY % formula       "%prefix;formula">
<!ENTITY % glosslist     "%prefix;glosslist">
<!ENTITY % i             "%prefix;i">
<!ENTITY % identification "%prefix;identification">
<!ENTITY % keyword       "%prefix;keyword">
<!ENTITY % label         "%prefix;label">
<!ENTITY % li            "%prefix;li">
<!ENTITY % moreinfo      "%prefix;moreinfo">
<!ENTITY % projectName    "%prefix;projectName">
<!ENTITY % footnote      "%prefix;footnote">
<!ENTITY % orderedlist   "%prefix;orderedlist">
<!ENTITY % p             "%prefix;p">
<!ENTITY % br            "%prefix;br">
<!ENTITY % participants  "%prefix;participants">
<!ENTITY % person        "%prefix;person">
<!ENTITY % raweb         "%prefix;raweb">
<!ENTITY % ref           "%prefix;ref">
<!ENTITY % refperson     "%prefix;refperson">
```

```
<!ENTITY % presentation  "%prefix;presentation">
<!ENTITY % fondements    "%prefix;fondements">
<!ENTITY % domaine       "%prefix;domaine">
<!ENTITY % logiciels     "%prefix;logiciels">
<!ENTITY % ressource     "%prefix;ressource">
<!ENTITY % resultats     "%prefix;resultats">
<!ENTITY % contrats      "%prefix;contrats">
<!ENTITY % international "%prefix;international">
<!ENTITY % diffusion     "%prefix;diffusion">
<!ENTITY % shortname     "%prefix;shortname">
<!ENTITY % simplelist    "%prefix;simplelist">
<!ENTITY % simplemath    "%prefix;simplemath">
<!ENTITY % small         "%prefix;small">
<!ENTITY % span          "%prefix;span">
<!ENTITY % sub           "%prefix;sub">
<!ENTITY % subsection    "%prefix;subsection">
<!ENTITY % sup           "%prefix;sup">
<!ENTITY % strong        "%prefix;strong">
<!ENTITY % table         "%prefix;table">
<!ENTITY % team          "%prefix;team">
<!ENTITY % term          "%prefix;term">
<!ENTITY % td            "%prefix;td">
<!ENTITY % th            "%prefix;th">
<!ENTITY % tr            "%prefix;tr">
<!ENTITY % tt            "%prefix;tt">
<!ENTITY % theme         "%prefix;theme">
<!ENTITY % bodyTitle     "%prefix;bodyTitle">
<!ENTITY % topic         "%prefix;topic">
<!ENTITY % UR            "%prefix;UR">
```

This list comes from a second file.

```
<!ENTITY % address       '%prefix;address'>
<!ENTITY % contact       '%prefix;contact'>
<!ENTITY % email         '%prefix;email'>
<!ENTITY % firstname     '%prefix;firstname'>
<!ENTITY % lastname      '%prefix;lastname'>
<!ENTITY % status        '%prefix;status'>
```

A decoration is either , <tt>, <i>, <b>, <big>, <small>, <sub>, <sup>, or . This corresponds to the <hi> element of the old DTD. A inline object is a decoration, or a <ressource>, <ref>, <code>, <anchor>, <formula>, <br>, , <LaTeX>, or <TeX>. A list is <glosslist>, <orderedlist>, <simplelist>, or a <descriptionlist>. A doc-block is a <table>, a list, or a <p>. Finally, a block is a doc-block, a <footnote> or a <object>.

```
<!ENTITY % decoration "%em; | %tt; | %i; | %b; | %big; | %small; | %sub; |
   %sup; | %strong; ">
<!ENTITY % inline "%decoration; | %ressource; | %ref; | %code; | %anchor; |
   %formula; | %br; | %span; | %LaTeX; | %TeX;" >
<!ENTITY % list "%glosslist; | %orderedlist; | %simplelist; | %descriptionlist;">
<!ENTITY % doc-block "%table; | %list; | %p;">
<!ENTITY % block  "%doc-block; | %footnote; | %object;">
```

We define here two entities; they say that the id attribute is required or optional. In any case it is an ID.

```
<!ENTITY % id   "id          ID      #IMPLIED">
<!ENTITY % id_r "id          ID      #REQUIRED">
```

We define an attribute that says that the language can be French or English. We declare also some attributes used for links.

```
<!ENTITY % xml-lang     'xml:lang   ( fr | en )             "en"        '>

<!ENTITY % xlink
        'xmlns:xlink    CDATA                   #FIXED "%XLINK.xmlns;"
         xlink:href     CDATA                   #REQUIRED
         xlink:type     CDATA                   #FIXED "simple"
         xlink:show     ( new | replace | embed) "replace"
         xlink:actuate  ( onLoad | onRequest)    "onRequest"'>
```

We declare here three files and include them. We have inlined the persons.mod file. We do not show the 'biblio.dtd' file; it is a copy of the TEI, and is rather long.

```
<!ENTITY % persons-mod  SYSTEM "persons.mod">
<!ENTITY % mathmlDTD    SYSTEM "mathml2.dtd">
<!ENTITY % biblioDTD    SYSTEM "biblio.dtd">
%mathmlDTD;
%biblioDTD;
%persons-mod;
```

We define here some abbreviations. A listSection defines the list of the eight standard sections: we have <presentation>, <fondements>, <domaine>, <logiciels>, <resultats>, <contrats>, <international>, <diffusion>. These are the same sections as in the old DTD. A contentSection describes what you can put in a section: an optional <bodyTitle> followed by some <block> or <subsection>.

```
<!ENTITY % listSection "(%presentation;), (%fondements;)?,
   (%domaine;)?, (%logiciels;)?, (%resultats;), (%contrats;)?,
   (%international;)?, (%diffusion;)?">
<!ENTITY % contentSection "(%bodyTitle;)?, ((%block;)* | (%subsection;)*) ">
```

Here are the sections. They all have id as required attribute.

```
<!ELEMENT %presentation; ( %contentSection; )>
<!ATTLIST %presentation; %id_r;>

<!ELEMENT %fondements; ( %contentSection; )>
<!ATTLIST %fondements; %id_r;>

<!ELEMENT %domaine; ( %contentSection; )>
<!ATTLIST %domaine; %id_r;>

<!ELEMENT %logiciels; ( %contentSection; )>
<!ATTLIST %logiciels; %id_r;>

<!ELEMENT %resultats; ( %contentSection; )>
<!ATTLIST %resultats; %id_r;>

<!ELEMENT %contrats; ( %contentSection; )>
<!ATTLIST %contrats; %id_r;>

<!ELEMENT %international; ( %contentSection; )>
<!ATTLIST %international; %id_r;>
```

```
475
476 <!ELEMENT %diffusion; ( %contentSection; )>
477 <!ATTLIST %diffusion; %id_r;>
```

We define identSection to be the identification part of a section; this is a list of <participants>, a list of <keyword> and an optional <moreinfo>.

```
478 <!ENTITY % identSection "( (%participants;)*, (%keyword;)*, (%moreinfo;)?)">
```

A <subsection> is formed of an optional <bodyTitle>, an identSection, followed by blocks or <subsection>. It can have a topic attribute. Note: only children of a section can have a topic.

```
479 <!ELEMENT %subsection; ( (%bodyTitle;)?, (%identSection;),
480     ((%block;) | (%subsection;))* ) >
481 <!ATTLIST %subsection;
482         %id;
483         topic   IDREF   #IMPLIED>
```

The <raweb> is formed of the eight standard sections, in %listSection; plus the bibliography, plus a <identification> part and a list of <topic> declarations.

```
484 <!ELEMENT %raweb; (%identification;, (%topic;)*, %listSection;, biblio) >
485 <!ATTLIST %raweb; year NMTOKEN #IMPLIED
486         %xml-lang;
487         xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink"
488         xmlns:html  CDATA #FIXED "http://www.w3.org/1999/xhtml"
489         %id;>
```

The <identification> element contains <shortname> <projectName>, <theme>, <team>, a sequence of <UR> and an optional <moreinfo>. It has a isproject attribute. A <topic> contains only text.

```
490 <!ELEMENT %identification; ( %shortname;, %projectName;, %theme;, %team;,
491       (%UR;)+, (%moreinfo;)?) >
492 <!ATTLIST %identification; isproject    (true|false) "true"  %id; >
493
494 <!ELEMENT %topic; (#PCDATA) >
495 <!ATTLIST %topic; %id_r;>
```

Trivial elements.

```
496 <!ELEMENT %theme; (#PCDATA)>
497 <!ATTLIST %theme;  id ID #IMPLIED>
498 <!ELEMENT %bodyTitle; (#PCDATA|%inline;)* >
499 <!ATTLIST %bodyTitle; id ID #IMPLIED>
500 <!ELEMENT %projectName; (#PCDATA|%inline;)* >
501 <!ATTLIST %projectName; id ID #IMPLIED>
502 <!ELEMENT %shortname; (#PCDATA|%inline;)* >
503 <!ATTLIST %shortname; id ID #IMPLIED>
```

An <UR> element is empty, but has a name attribute that identifies it.

```
504 <!ENTITY % listUR "Sophia|Rocquencourt|RhoneAlpes|Rennes|Lorraine|Futurs">
505 <!ELEMENT %UR; EMPTY>
506 <!ATTLIST %UR; name (%listUR;) #REQUIRED >
507 <!ATTLIST %UR; id ID #IMPLIED>
```

A <moreinfo> element can contain block, inline objects, footnotes.

```
508 <!ELEMENT %moreinfo; (#PCDATA | %doc-block; | %footnote; | %inline;)* >
509 <!ATTLIST %moreinfo; %id;>
```

A <caption> contains inline objects.

```
<!ELEMENT %caption; (#PCDATA|%inline;)* >
<!ATTLIST %caption; id ID #IMPLIED>
```

A element contains inline stuff. It has a style attribute that explains how it can be emphasized.

```
<!ENTITY % styles "highlight | underline">
<!ELEMENT %em; (#PCDATA | %inline;)*>
<!ATTLIST %em; style (%styles;) #IMPLIED>
<!ATTLIST %em; id ID #IMPLIED>
```

We define here some elements: , <i>, <tt>, <b>, <big>, <small>, <sub>, and <sup>. They contain inline stuff.

```
<!ELEMENT %strong;  (#PCDATA | %inline;)*>
<!ATTLIST %strong;  id ID #IMPLIED>
<!ELEMENT %i;       (#PCDATA | %inline;)*>
<!ATTLIST %i;       id ID #IMPLIED>
<!ELEMENT %tt;      (#PCDATA | %inline;)*>
<!ATTLIST %tt;      id ID #IMPLIED>
<!ELEMENT %b;       (#PCDATA | %inline;)*>
<!ATTLIST %b;       id ID #IMPLIED>
<!ELEMENT %big;     (#PCDATA | %inline;)*>
<!ATTLIST %big;     id ID #IMPLIED>
<!ELEMENT %small;   (#PCDATA | %inline;)*>
<!ATTLIST %small;   id ID #IMPLIED>
<!ELEMENT %sub;     (#PCDATA | %inline;)*>
<!ATTLIST %sub;     id ID #IMPLIED>
<!ELEMENT %sup;     (#PCDATA | %inline;)*>
<!ATTLIST %sup;     id ID #IMPLIED>
```

I don't understand this.

```
<!ELEMENT %span; (#PCDATA | %inline;)*>
<!ATTLIST %span; align (left|center|right) "left" id ID #IMPLIED >
```

The <code> element is unused by the Raweb, as well as <anchor>.

```
<!ELEMENT %code;   (#PCDATA | %inline;)*>
<!ELEMENT %anchor; EMPTY>
<!ATTLIST %anchor; %id_r;>
```

A <ref> element is a link. It can contain text. It has attributes.

```
<!ELEMENT %ref; (#PCDATA | %inline;)* >
<!ATTLIST %ref;
        location (intern | biblio | extern) "extern"
        %xlink; >
<!ATTLIST %ref; id ID #IMPLIED>
```

A <keyword> contains only characters (what about "λ-calculus" ?). The theme attribute is currently unused.

```
<!ELEMENT %keyword; (#PCDATA) >
<!ATTLIST %keyword; %id; theme CDATA #IMPLIED>
```

A <footnote> can contain arbitrary text (well, there are restrictions...). The place attribute is not used.

```
<!ELEMENT %footnote; (#PCDATA | %block;)* >
<!ATTLIST %footnote; %id; place CDATA "unspecified">
```

We have four types of lists. They contain all some `<li>` children; in the case of a glossary or description, we can have `<label>`. The purpose of the title attribute is unclear.

```
<!ELEMENT %simplelist; (%li;)+ >
<!ATTLIST %simplelist; %id; title CDATA #IMPLIED>

<!ELEMENT %orderedlist; (%li;)+ >
<!ATTLIST %orderedlist;  %id; title CDATA #IMPLIED>

<!ELEMENT %glosslist; (%label; , %li;)+ >
<!ATTLIST %glosslist; %id; title CDATA #IMPLIED>

<!ELEMENT %descriptionlist; (%label; | %li;)+ >
<!ATTLIST %descriptionlist; %id; title CDATA #IMPLIED>
```

A `<label>` in a list contains only inline elements, while a `<li>` can be more complicated.

```
<!ELEMENT %label; (#PCDATA|%inline;)* >
<!ATTLIST %label; %id;>

<!ELEMENT %li; (#PCDATA | %doc-block; | %inline;)* >
<!ATTLIST %li; %id;>
```

A `<table>` contains some `<tr>` and an optional `<caption>`. It can have a title attribute. A `<tr>` contains some `<th>` and `<td>`. These two elements have the same structure. However, a `<td>` element can have a style attribute.[2]

```
<!ELEMENT %table; ((%tr;)+, (%caption;)?) >
<!ATTLIST %table;  %id; title CDATA #IMPLIED >
<!ELEMENT %tr; (%th; | %td;)+>
<!ATTLIST %tr; %id; >
<!ELEMENT %td; (#PCDATA | %doc-block; | %inline;)*>
<!ATTLIST %td; style CDATA #IMPLIED %id; >
<!ELEMENT %th; (#PCDATA | %doc-block; | %inline;)*>
<!ATTLIST %th; %id; >
```

This defines a `<p>` element.

```
<!ELEMENT %p; (#PCDATA | %inline; | %footnote;)*>
<!ATTLIST %p; %id;
              noindent CDATA #IMPLIED
              rend CDATA #IMPLIED>
```

A `<pers>` is formed of an optional `<firstname>`, `<lastname>`, an optional `<contact>`, an optional `<status>`, and an optional `<moreinfo>`. .

```
<!ELEMENT %person; ( (%firstname;)?, %lastname;, (%contact;)?,
   (%status;)?, (%moreinfo;)?) >
<!ATTLIST %person;       %id;>
```

The `<firstname>` and `<lastname>` elements, as well as `<email>`, contain characters. The content of `<contact>` is email or address; the adress is defined by the TEI[3].

```
<!ELEMENT %firstname; (#PCDATA) >
<!ATTLIST %firstname; %id;>
<!ELEMENT %lastname; (#PCDATA) >
<!ATTLIST %lastname; %id;>
```

[2] How is this attribute defined?

[3] It is too complicated to explain here

```
<!ELEMENT %contact; (%email; | %address;)* >
<!ELEMENT %email; (#PCDATA) >
```

The <status> element contains an optional <moreinfo>. Its type attribute is an element of list not shown here (I don't understand the purpose of the list).

```
<!ENTITY % listStatus "Professor... None">
<!ELEMENT %status; (%moreinfo;)? >
<!ATTLIST %status; type (%listStatus;) #IMPLIED >
```

The <team> section contains some <participants> elements, and an optional <moreinfo>.

```
<!ELEMENT %team; ( (%participants;)+, (%moreinfo;)?)>
<!ATTLIST %team; %id;>
```

A <participants> element contains some <person> or <refperson> elements, followed by an optional <moreinfo>.

```
<!ELEMENT %participants; ( (%person; | %refperson;)+, (%moreinfo;)?)>
<!ATTLIST %participants; category  CDATA #IMPLIED id ID #IMPLIED>
```

A <refperson> is a link to a <person>.

```
<!ELEMENT %refperson; EMPTY>
<!ATTLIST %refperson; ref IDREF #REQUIRED>
```

This defines an <object> and a <ressource>. An object can be used to include a floating figure

```
<!ELEMENT %object; ( (%bodyTitle;)?, (%table;)+, (%caption;)? ) >
<!ATTLIST %object; %id_r;>
<!ELEMENT %ressource; ( (%caption;)? ) >
<!ATTLIST %ressource;
        %id;
        %xlink;
        media (WEB | PRINT ) "WEB"
        width NMTOKEN #IMPLIED
        height NMTOKEN #IMPLIED
        preview CDATA #IMPLIED
        type (inline|display) "inline"
        framed CDATA #IMPLIED
        scale CDATA #IMPLIED
        angle CDATA #IMPLIED>
```

Math formulas are defined like in the old DTD.

```
<!ELEMENT %formula;  (%simplemath; | %math.qname;) >
<!ATTLIST %formula;  %id;  type (inline|display) "inline" >
<!ELEMENT %simplemath; (#PCDATA | %inline;)* >
<!ATTLIST %simplemath;  %id;  type (inline|display) "inline" >
```

Index

In this index, the page number of a command refers often to the start of the verbatim block where it is used or defined. In many cases, you will find the command one page after the number given in the index.

Bibliography

[1] David Carlisle. XMLTEX: A non validating (and not 100% conforming) namespace aware XML parser inplemented in TeX. *TUGboat*, 21(3):193–199, 2000.

[2] David Carlisle, Patrick Ion, Robert Miner, and Nico Poppelier (editors). Mathematical Markup Language (MathML) Version 2.0. http://www.w3.org/TR/MathML2/, 2001.

[3] Consortium WWW. La spécification xml. *Cahier Gutenberg*, (33-34), 1999. This is a French translation of the specifications, http://www.w3.org/TR/1998/REC-xml-19980210.

[4] Michael Kay. *XSLT, Programmer's Reference*. Wrox Press Ltd, 2nd edition, 2001.

[5] Donald E. Knuth. *The TeXbook*. Addison Wesley, 1984.

[6] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, and Chris Rowley. *The LaTeX companion, second edition*. Addison Wesley, 2004.

[7] The Unicode Consortium. *The Unicode Standard, version 4.0*. Addison Wesley, 2003.

[8] W3C. Extensible Markup Language (XML) 1.0 (Third Edition). http://www.w3.org/TR/REC-xml/, 1998. Third edition published in 2004.

[9] W3C. Extensible Markup Language (XML) 1.1. http://www.w3.org/TR/xml11/, 2004.

[10] Web consortium. Namespaces in XML. http://www.w3.org/TR/1999/REC-xml-names-19990114, 1999.

Contents

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803